UM ALGORITMO EXATO PARA A OTIMIZAÇÃO DE CARTEIRAS DE INVESTIMENTO COM RESTRIÇÕES DE CARDINALIDADE

Dissertação de mestrado em matemática aplicada financiada pelo CNPq

IMECC - UNICAMP

Pedro Ferraz Villela

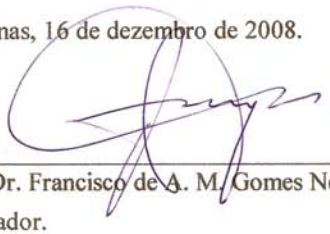
Prof. Dr. Francisco de Assis Magalhães Gomes Neto - Orientador

Dezembro de 2008

UM ALGORITMO EXATO PARA A OTIMIZAÇÃO DE CARTEIRAS DE INVESTIMENTO COM RESTRIÇÕES DE CARDINALIDADE

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por **Pedro Ferraz Villela** e aprovada pela comissão julgadora.

Campinas, 16 de dezembro de 2008.



Prof. Dr. Francisco de A. M. Gomes Neto
Orientador.

Banca Examinadora:
Prof. Dr. Francisco de Assis Magalhães Gomes Neto (IMECC – UNICAMP)
Prof. Dr. Antonio Carlos Moretti (IMECC – UNICAMP)
Prof. Dr. Paulo Augusto Valente Ferreira (FEEC – UNICAMP)

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica, UNICAMP, como requisito parcial para obtenção do Título de MESTRE em Matemática Aplicada.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Crisllene Queiroz Custódio – CRB8a 162/2005

Villela, Pedro Ferraz
V715a Um algoritmo exato para a otimização de carteiras de investimento com restrições de cardinalidade / Pedro Ferraz Villela -- Campinas, [S.P. : s.n.], 2008.

Orientador : Francisco de Assis Magalhães Gomes Neto
Dissertação (Mestrado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Otimização de Carteiras de Investimento. 2. Método de Lemke. 3. Algoritmo Branch and Bound. 4. Restrições de cardinalidade. I. Gomes Neto, Francisco de Assis Magalhães. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

(cqc/imecc)

Título em inglês: An exact algorithm for portifolio optimization with cardinality constraints.

Palavras-chave em inglês (Keywords): 1. Portfolio optimization. 2. Lemke's method. 3. Branch and Bound algorithms. 4. Cardinality constraints.

Área de concentração: Otimização

Titulação: Mestre em Matemática Aplicada

Banca examinadora: Prof. Dr. Francisco de Assis Magalhães Gomes Neto (IMECC-UNICAMP)
Prof. Dr. Antonio Carlos Moretti (IMECC-UNICAMP)
Prof. Dr. Paulo Augusto Valente Ferreira (FEEC-UNICAMP)

Data da defesa: 16/12/2008

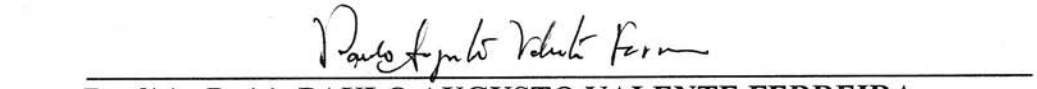
Programa de Pós-Graduação: Mestrado em Matemática Aplicada

Dissertação de Mestrado defendida em 16 de dezembro de 2008 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.

Prof(a). Dr(a). FRANCISCO DE ASSIS MAGALHÃES GOMES NETO

Prof(a). Dr(a). ANTONIO CARLOS MORETTI



Prof(a). Dr(a). PAULO AUGUSTO VALENTE FERREIRA

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer ao meu orientador e amigo Chico, que sempre esteve junto comigo durante todo o trabalho. Agradeço por toda a sua disposição e paciência ao longo desse período

Também agradeço muito a minha família, que sempre esteve comigo nos momentos mais difíceis, apoiando-me com muito amor. Em especial aos meus pais, que se dispuseram a me ajudar até nos horários mais indejesáveis.

Sou muito grato também aos meus amigos Carlos e Rodrigo, que sempre se disporam a me ajudar quando preciso. Em especial ao Carlos, pela ajuda vital na formatação do texto.

Por último, agradeço ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), pelo essencial apoio financeiro prestado durante a vigência do trabalho.

RESUMO

Neste trabalho, propomos um método exato para a resolução de problemas de programação quadrática que envolvem restrições de cardinalidade. Como aplicação, empregamos o método para a obtenção da fronteira eficiente de um problema (bi-objetivo) de otimização de carteiras de investimento. Nosso algoritmo é baseado no método *Branch-and-Bound*. A chave de seu sucesso, entretanto, reside no uso do método de Lemke, que é aplicado para a resolução dos subproblemas associados aos nós da árvore gerada pelo *Branch-and-Bound*. Ao longo do texto, algumas heurísticas também são introduzidas, com o propósito de acelerar a convergência do método. Os resultados computacionais obtidos comprovam que o algoritmo proposto é eficiente.

Palavras-chave: Otimização de carteiras de investimento. método de Lemke. *Branch-and-Bound*. Restrição de cardinalidade.

ABSTRACT

In this work, we propose an exact method for the resolution of quadratic programming problems involving cardinality restrictions. As an application, the algorithm is used to generate the effective Pareto frontier of a (bi-objective) *portfolio* optimization problem. This algorithm is based on the *Branch-and-Bound* method. The key to its success, however, resides in the application of Lemke's method to the resolution of the subproblems associated to the nodes of the tree generated by the *Branch-and-Bound* algorithm. Throughout the text, some heuristics are also introduced as a way to accelerate the performance of the method. The computational results acquired show that the proposed algorithm is efficient.

Keywords: *Portfolio* optimization. Lemke's method. Cardinality constraints. *Branch-and-Bound* algorithm

SUMÁRIO

INTRODUÇÃO

Esta dissertação trata do problema de otimização de uma carteira de investimento, ou *portfolio*, que é o conjunto de ativos financeiros nos quais se investe. Em geral, ao montar uma carteira de investimento, o investidor procura maximizar o lucro, bem como minimizar o risco de perda de capital. Para lidar com o problema, o investidor precisa utilizar um modelo matemático que leve em conta esses dois objetivos antagônicos.

Uma das primeiras formulações matemáticas para esse problema de investimento foi o modelo de Média-Variância proposto por Markowitz [13] nos anos 50 do século 20. Ainda hoje, esse modelo tem uma boa aceitação por parte dos investidores. Em sua forma usual, o modelo de Markowitz se resume a um problema de programação quadrática no qual a função objetivo representa o risco a ser minimizado e o retorno mínimo da carteira é descrito através de uma restrição linear.

Com o passar do tempo, várias alterações foram sugeridas com o propósito de tornar o modelo de Markowitz mais compatível com o mercado financeiro real. Dentre essas alterações, destacamos a inclusão de restrições que limitam o tamanho da carteira e de restrições que estipulam um limite inferior para o montante aplicado em cada ativo.

Com essas novas restrições, o problema de programação quadrática passa a envolver variáveis inteiras, além das variáveis reais que já eram usadas para representar o valor investido em cada ativo. Naturalmente, com a inclusão de variáveis inteiras, o problema se torna NP-completo, exigindo um grande esforço computacional para a obtenção da solução ótima.

Em virtude da dificuldade do problema, uma alternativa prática para a obtenção de soluções satisfatórias em um tempo aceitável é a adoção de métodos heurísticos. Dentre os

trabalhos desta classe, podemos citar o artigo de Chang *et al.* [5] que investiga o uso de algoritmos genéticos, da busca tabu e da têmpera simulada para a determinação do *portfolio* apropriado para cada perfil de investidor, contemplando desde os mais conservadores até os mais arrojados. Seguindo o trabalho de Chang *et al.*, Dias [7] propôs, recentemente, um novo algoritmo genético eficiente para resolver problemas de investimento.

Outra linha de solução do problema de carteiras de ativos financeiros envolve o emprego de métodos exatos, geralmente baseados no algoritmo *Branch-and-Bound* aliado a algum método para resolver problemas de otimização que envolvem variáveis contínuas. Dentre os trabalhos desta linha, podemos citar o desenvolvido por Bienstock [4], que utiliza técnicas de planos de corte aliadas ao algoritmo *Branch-and-Bound*. Outro trabalho igualmente importante foi o de Bertsimas e Shioda [3], que utiliza o algoritmo *Branch-and-Bound* aliado ao método de Lemke [6, 11].

Em nossa pesquisa, também desenvolvemos um algoritmo exato para o problema de *portfolios*, combinando o algoritmo *Branch-and-Bound* com o método de Lemke. Nosso objetivo principal é a obtenção de um algoritmo eficiente do ponto de vista computacional.

Esta dissertação está dividida da seguinte forma: no primeiro capítulo, fazemos uma introdução aos modelos de carteira de investimento, bem como apresentamos o tipo de problema problema específico que nos propomos a resolver. O Capítulo 2 é dedicado à apresentação detalhada do método de Lemke. No Capítulo 3, apresentamos o algoritmo *Branch-and-Bound* tradicional, seguido da versão implícita que desenvolvemos. No Capítulo 4, tratamos dos detalhes de implementação do *Branch-and-Bound* e do método de Lemke. Finalmente, o Capítulo 5 apresenta os resultados obtidos com a aplicação do nosso algoritmo a problemas reais de carteiras de investimento, que comprovam a eficiência do método que desenvolvemos.

CAPÍTULO 1

MODELOS DE CARTEIRAS DE INVESTIMENTO

1.1 O MODELO DE MARKOWITZ

O tipo de problema de *portfolio* a ser estudado nesse trabalho caracteriza-se pela escolha de como investir o capital disponível entre as várias ações existentes no mercado. A estratégia envolvida na resolução desse tipo de problema envolve tanto a maximização do lucro do investimento, como a minimização do risco de perda de capital em virtude das flutuações dos preços do mercado. Naturalmente, estes objetivos são contraditórios e cabe ao investidor decidir como combiná-los.

O modelo matemático clássico para esse tipo de problema de *portfolio* é baseado na idéias propostas por Markowitz [13] em 1952. Definidos os N diferentes ativos que compõem a carteira de investimento, o montante a ser investido em cada ativo pode ser obtido resolvendo-se o problema de programação quadrática

$$\begin{aligned} \min \quad & x^T Q x \\ s.a \quad & \mu^T x = \rho \\ & \sum_{i=1}^{N} x_i = 1 \\ & x_i \geq 0, \quad i = 1,...,N. \end{aligned} \quad , \tag{1.1}$$

onde

- $Q \in \Re^{NxN}$ é a matriz de covariância do problema. Assim, o elemento q_{ij} da matriz representa a covariância do ativo *i* em relação ao ativo *j*. Além disso, a matriz *Q* é simétrica e semidefinida positiva ($Q = Q^{\mathrm{T}} \geq 0$).
- $x \in \Re^N$ é o vetor de incógnitas, que indica como o montante total disponível deve ser distribuído entre os ativos. Desta forma, a componente x_i fornece a fração do montante a ser investida no ativo *i*.
- $\mu \in \Re^N$ é o vetor de retornos esperados. Ou seja, o elemento μ_i representa o retorno esperado para o ativo *i*.
- $\rho \in \Re$ é o nível desejado de retorno do *portfolio*.

Nesse modelo, temos como objetivo minimizar o risco, representado pela função quadrática baseada na matriz de covariância, para um nível predefinido de retorno, definido pela primeira restrição. A segunda restrição indica apenas que desejamos gastar todo o montante disponível, enquanto a terceira é usada para evitar que a porcentagem do dinheiro gasta em cada ativo seja negativa, o que significa que não permitimos a venda de ativos. Os conceitos de risco e retorno serão explicados detalhadamente na Seção 1.4.

Para definir um problema específico na forma (1.1), faz-se uma análise histórica dos ativos financeiros escolhidos para compor a carteira, de modo que seja possível criar tanto uma medida de risco (em geral, a variância), como também obter o valor esperado de retorno de capital (esperança).

1.2 OUTROS MODELOS

Naturalmente, o Modelo (1.1) poderia ser substituído por outro no qual se estabelecesse um nível para a variância e o retorno fosse minimizado. Outra possibilidade seria a adoção de uma desigualdade na restrição de retorno esperado, de modo que ρ correspondesse ao retorno mínimo admissível. Ao longo dessa seção, apresentaremos outros modelos que podem ser obtidos a partir de (1.1).

1.2.1 O modelo paramétrico

Uma modificação possível do Modelo (1.1) corresponde a incluir, além da função de risco, o desejo de maximizar o retorno dentro da função objetivo. Para tanto, é preciso utilizar um parâmetro λ que que pondera os objetivos conflitantes de redução do risco e aumento do retorno. Neste caso, obtemos o problema abaixo, no qual temos apenas as restrições de uso de todo o montante disponível e de não negatividade das variáveis.

$$\begin{aligned} \min \quad & \lambda x^T Q x - (1-\lambda)\mu^T x \\ s.a \quad & \sum_{i=1}^{N} x_i = 1 \\ & x_i \geq 0, i = 1,..., N. \end{aligned} \tag{1.2}$$

Tomando $0 \leq \lambda \leq 1$, observamos que para $\lambda = 0$, apenas o retorno é considerado, enquanto $\lambda = 1$ corresponde à simples minimização do risco.

Caso a matriz Q seja positiva definida, o problema é convexo e possui solução única, pois o único mínimizador local existente é também minimizador global.

1.2.2 O modelo MAD

Problemas de programação quadrática como aqueles apresentados em (1.1) e (1.2) não são os únicos modelos utilizados na otimização de *portfolios*. O modelo do desvio absoluto com relação à média (MAD) [14], por exemplo, tem como idéia a minimização do risco através de uma função objetivo linear.

Neste modelo, considera-se a existência de T períodos para os quais conhecemos o retorno dos ativos disponíveis para aplicação. O objetivo do modelo é minimizar a soma do desvio absoluto do retorno de cada *portfolio* em relação à média de retorno para todos os períodos. Em termos matemáticos, a função a ser minimizada tem a forma

$$f(x) = \frac{1}{T}\sum_{t=1}^{T}\left|\sum_{i=1}^{N}(r_{i,t} - \mu_i)x_i\right|,$$

onde $r_{i,t}$ é o retorno de um determinado ativo i no período de tempo t.

Para converter o modelo em um problema de programação linear, usamos variáveis auxiliares m_t, de modo que cada termo que envolve o valor absoluto dá origem a duas restrições de desigualdade. Considerando também as restrições do problema de Markowitz, chegamos ao seguinte problema:

$$
\begin{aligned}
\min \quad & \frac{1}{T}\sum_{t=1}^{T} m_t \\
s.a \quad & \sum_{i=1}^{N}(r_{i,t}-\mu_i)x_i \le m_t, \quad t=1,..,T \\
& \sum_{i=1}^{N}(r_{i,t}-\mu_i)x_i \ge -m_t, \quad t=1,..,T \\
& \mu^T x = \rho \\
& \sum_{i=1}^{N} x_i = 1 \\
& m_t \ge 0, \quad t=1,..,T \\
& x_i \ge 0, \quad i=1,..,N.
\end{aligned}
$$

1.2.3 Um modelo de programação por metas

Outra abordagem linear interessante é a que envolve programação por metas (*goal programming*) [14]. Neste caso, minimiza-se tanto a chance de um retorno negativo como a de um risco elevado, dando um peso específico a cada uma dessas possibilidades. Assim, caso o investidor seja arrojado, o peso sobre o retorno será maior, enquanto o investidor conservador irá optar por usar um peso maior sobre o risco.

Para esse modelo, utilizamos os seguintes parâmetros e variáveis

Parâmetros:

- W_1: penalidade positiva associada ao retorno abaixo do objetivo esperado.
- W_2: penalidade positiva associada ao risco excessivo do *portfolio* em relação ao valor esperado.
- $Risk_p$: risco global associado ao *portfolio* definido pelo investidor.
- $Risk_i$: risco associado ao ativo *i* definido pelo investidor.

Variáveis:

- n_1: desvio negativo em relação ao retorno esperado.
- p_1: desvio positivo em relação ao retorno esperado.
- n_2: denota o desvio negativo em relação ao nível de risco.
- p_2: denota o desvio positivo em relação ao nível de risco.

Dessa forma, temos o seguinte modelo:

$$
\begin{aligned}
\min \quad & W_1 n_1 + W_2 p_2 \\
s.a \quad & \sum_{i=1}^{N} x_i \mu_i + n_1 - p_1 = \rho \\
& \sum_{i=1}^{N} Risk_i x_i + n_2 - p_2 = Risk_p \\
& \sum_{i=1}^{N} x_i = 1 \\
& n_1, n_2, p_1, p_2 \geq 0 \\
& x_i \geq 0, i = 1,2,...,N.
\end{aligned}
$$

A função objetivo procura minimizar apenas o risco acima da meta e o retorno abaixo da meta estabelecida. Deve-se notar que as variáveis n_1 e p_1 não serão diferentes de zero ao mesmo tempo, o mesmo ocorrendo com n_2 e p_2,.

1.3 O MODELO COM RESTRIÇÃO DE CARDINALIDADE E CANALIZAÇÕES.

Outra linha de desenvolvimento de modelos para a otimização de carteiras de investimentos está relacionada à inclusão de restrições que tornam o problema mais compatível com as situações encontradas na vida real. Nessa linha, destacamos o modelo no qual se restringe a quantidade de ativos que irão compor a carteira, uma vez que normalmente não é viável investir

em todos os N ativos disponíveis no mercado. Este modelo geralmente inclui também um conjunto de limites inferiores, l_i, e superiores, u_i, para as frações aplicadas a cada ativo selecionado.

Para restringir a K o número de ativos, incluímos no modelo as variáveis binárias auxiliares δ_i, $i = 1,...,N$, de modo que $\delta_i = 1$ caso o ativo i faça parte da carteira e $\delta_i = 0$ caso contrário.

Já com relação aos limites inferiores e superiores, como estes só são aplicados às variáveis efetivamente incluídas na carteira, em lugar de utilizarmos canalizações simples, recorremos a desigualdades mistas, ou seja, com variáveis inteiras e reais.

Tomando por base o modelo de Markowitz, obtemos o seguinte problema

$$
\begin{aligned}
\min \quad & x^T Q x \\
s.a \quad & \mu^T x = \rho \\
& \sum_{i=1}^{N} x_i = 1 \\
& l_i \delta_i \leq x_i \leq u_i \delta_i, \quad i = 1,..., N \\
& \delta_i \in \{0,1\}, \quad i = 1,..., N \\
& \sum_{i=1}^{N} \delta_i \leq K, \quad K \leq N \\
& x_i \geq 0, \quad i = 1,..., N.
\end{aligned}
\tag{1.3}
$$

Em (1.3), se δ_i vale 0, x_i também será igual a 0. Por outro lado, se $\delta_i = 1$, temos $l_i \leq x_i \leq u_i$ Como temos no máximo K variáveis inteiras δ_i iguais a 1, somos forçados a escolher no máximo K ativos.

Naturalmente, a restrição de cardinalidade pode ser incluída em todos os problemas apresentados neste capítulo, bem como em muitos outros modelos para problemas de otimização de carteiras. Para maiores informações, sugere-se consultar o artigo de Mitra *et al.*[14].

1.4 MEDIDAS DE RETORNO E RISCO

Naturalmente, não é possível medir o retorno futuro de um ativo. Assim, para estimar o retorno esperado, fazemos uma análise histórica, na esperança de que os valores registrados no passado recente voltem a ocorrer. Em termos matemáticos, supondo que um ativo tenho assumido historicamente *n* valores diferentes, definindo x_i como o i-ésimo retorno possível e p_i como a respespectiva probabilidade de ocorrência, uma maneira de representar o retorno esperado do ativo é

$$\mu = \bar{x} = E(x) = \sum_{i=1}^{n} x_i p_i \ .$$

O retorno esperado pode ser visto também como a média dos possíveis valores que o ativo assumiu historicamente.

Já o conceito de risco é algo muito mais complexo pois não existe uma fórmula universal para se medir o risco de um investimento, uma vez que o conceito é extremamente vasto e abstrato, não podendo ser representado de maneira exata. A medida de risco mais usada e intuitiva é a variância, como visto no modelo de Markowitz. Porém, esta é apenas uma das possíveis representações do risco.

Uma maneira mais geral de representar o risco envolve o uso do que chamamos de momentos parciais [14], ou simplesmente LPM (abreviatura do termo inglês *lower partial moments*), apresentados a seguir.

Seja α o parâmetro que indica o momento do retorno da distribuição, τ o nível pré-definido de retorno do investimento, e $F(x)$ a função de densidade de probabilidade do investimento com retorno x. Neste caso, definimos os momentos parciais de ordem α como

$$F_\alpha(\tau) = LPM_\alpha(\tau, x) = \int_{-\infty}^{\tau} (\tau - x)^\alpha f(x)\, dx = E\left\{ \left(\max[0, \tau - x]\right)^\alpha \right\}, \alpha > 0 \, .$$

O LPM pode lidar tanto com medidas simétricas quanto assimétricas de risco. No caso simétrico, as medidas de risco incluem, por exemplo, a variância e o desvio absoluto com relação à média (MAD), como vemos abaixo.

- Variância: definindo o nível de retorno e o momento como $\tau = \bar{x}$ e $\alpha = 2$, obtemos

$$\sigma^2 = LPM_2(\bar{x}, x) = \int_{-\infty}^{+\infty} (\bar{x} - x)^2 f(x)\, dx = E\left\{(\bar{x} - x)^2\right\}.$$

- MAD: definindo $\tau = \bar{x}$ e $\alpha = 1$, obtemos

$$MAD = LPM_1(\bar{x}, x) = \int_{-\infty}^{+\infty} \left|\bar{x} - x\right| f(x)\, dx = E\left\{\left|\bar{x} - x\right|\right\}.$$

No caso assimétrico, desprezamos da medida de risco o retorno acima do nível desejado. Dentre as medidas assimétricas de risco, destacamos a semivariância e o MAD^-, que são obtidas a partir das definições acima, substituindo o limite superior da integral por $\bar{x}$, em lugar de $+\infty$. Outra medida assimétrica muito usada atualmente é denominada Valor-em-Risco, ou VaR (do inglês *Value-at-Risk*).

Embora a escolha da medida de risco seja um assunto em grande ebulição, com defensores arraigados para os vários modelos existentes, utilizaremos, neste trabalho, a medida clássica, ou seja, a variância, que ainda conta com o maior número de adeptos.

1.5 A FRONTEIRA EFICIENTE

Problemas de programação multi-objetivo, como o que temos, geralmente não possuem uma solução ótima única. Em outras palavras, não é razoável esperar que a carteira com o maior retorno seja também aquela que tem o menor risco. Para comparar soluções diferentes, usamos a noção de *dominância*.

Dizemos que uma solução x_1 domina uma outra solução x_2 se:

- $\forall i : f_i(x_1) \leq f_i(x_2)$;
- $\exists i : f_i(x_1) < f_i(x_2)$.

No caso de aplicações financeiras, dizemos que um *portfolio*, π_1, domina outro, π_2, se π_1 possui risco menor e retorno maior que π_2, podendo haver igualdade em uma das duas medidas.

Para um problema com p objetivos, a *fronteira eficiente de investimento*, ou *fronteira de Pareto*, é a curva em $\Re^p$ que representa o conjunto de pontos não dominados. Para o problema de carteiras de investimentos, a fronteira eficiente é a curva não decrescente que contém as melhores possibilidades de conciliação entre risco e retorno. Vale notar que, caso não tenhamos restrições de cardinalidade, a curva é "bem comportada", pois ela é não decrescente e possui uma reta suporte.

A Figura 1.1, mostrada a seguir, apresenta uma fronteira eficiente de investimento gerada a partir de ativos da bolsa de valores do Reino Unido. Nesta figura, o eixo das abscissas representa a medida de risco do *portfolio*, que nesse caso é a variância. Já o eixo das ordenadas representa o retorno médio dos ativos.

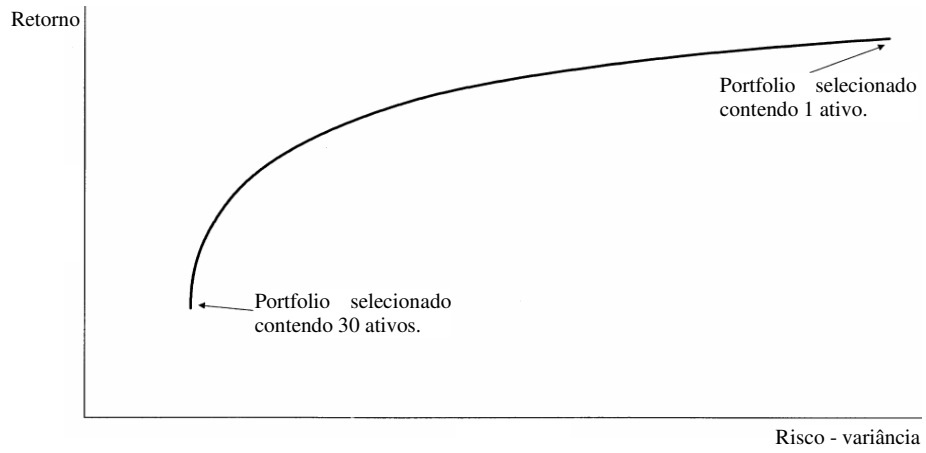


Figura 1.1 – FEI. dos ativos da bolsa de valores do Reino Unido (UK FTSE), extraída de [5].

A análise da Figura 1.1 revela que caso o investidor tenha uma grande aversão ao risco, ele deve optar por investir em um *portfolio* que tenha vários ativos, pois a possibilidade de perder dinheiro em todos eles é menor. Caso o investidor seja mais audaz e escolha priorizar o retorno, ele deve optar por investir em poucos ativos, em geral aqueles que apresentam o maior retorno, ainda que o risco de perda de capital seja grande devido à volatilidade do mercado financeiro. Vale notar ainda, que nenhuma das soluções da fronteira eficiente é "melhor" que as outras: cada ponto dela representa a melhor solução para o nível de risco especificado pelo investidor. Para maiores informações sobre fronteira eficiente de investimentos sugere-se consultar o livro de Luenberger [12].

Um modelo muito utilizado para a determinação da fronteira eficiente de investimento é o modelo parametrizado (1.2). Como a função objetivo desse problema é convexa, todo mínimo local é também mínimo global. Logo, cada ponto da fronteira nada mais é que a solução de (1.2) para um ou mais valores de λ.

Valores do parâmetro λ próximos de zero significam que o investidor prioriza o retorno, pouco se importando com o risco. Já valores de λ próximos de um, significam que o investidor prioriza excessivamente o risco, deixando o retorno em segundo plano. Logo, o parâmetro λ desempenha um papel essencial no modelo, uma vez que ele mede o nível de risco escolhido pelo investidor.

Na prática, uma maneira de se construir a Fronteira Eficiente envolve a discretização dos valores λ no intervalo [0,1] e a resolução do Problema (1.2) para cada um dos valores de λ selecionados. Em geral, essa discretização é feita definindo E como o tamanho da partição e tomando $\lambda \in \left\{0, \frac{1}{E-1}, \frac{2}{E-1}, \cdots, \frac{E-1}{E-1}\right\}$.

Quando se incorpora a restrição de cardinalidade, a fronteira eficiente passa a ser descontínua. Assim, a complexidade de problemas como o (1.4), a ser apresentado na próxima seção, é muito maior do que a de (1.2). Para maiores informações sobre a fronteira eficiente em problemas que envolvem restrições de cardinalidade pode-se consultar Chang *et al.* [5] e Dias [7].

1.6 PROBLEMAS DE PROGRAMAÇÃO QUADRÁTICA MISTA

Neste trabalho, nos concentramos no modelo apresentado na Seção 1.3, que inclui uma restrição de cardinalidade e limites inferiores para os montantes a serem aplicados nos ativos da carteira. Em nosso problema, não incluímos as restrições de limite superior para as variáveis.

Para permitir um maior controle sobre a minimização do risco e a maximização do retorno, combinamos o Modelo (1.3) com o modelo paramétrico, acrescentando o retorno parametrizado à função objetivo. Além disso, para facilitar a resolução do problema, transformamos em desigualdade a restrição de montante aplicado. Obtemos, assim, o problema

$$
\begin{aligned}
\min \quad & \lambda x^T Q x - (1-\lambda)\mu^T x \\
s.a \quad & \sum_{i=1}^{N} x_i \leq 1 \\
& x_i \geq l_i \, \delta_i, \; i = 1,..., N \\
& \delta_i \in \{0,1\}, \; i = 1,..., N \\
& \sum_{i=1}^{N} \delta_i \leq K, \; K \leq N \\
& x_i \geq 0, \; i = 1,..., N.
\end{aligned}
\tag{1.4}
$$

Com a conversão da restrição de montante aplicado em uma desigualdade, é possível que uma parte do dinheiro disponível não seja investida, particularmente no caso em que λ está próximo de 1, ou seja, quando a aversão ao risco é grande. Em particular, quando $\lambda = 1$, caso a matriz Q seja definida positiva, nada é investido e, neste caso, a função objetivo atingirá o mínimo apenas para $x = 0$.

Como não é razoável supor que um investidor mantenha em caixa grande parte de seu dinheiro, podemos considerar a existência de um ativo livre de risco (como a caderneta de poupança ou as letras do tesouro americano) que absorva o montante não aplicado nos ativos passíveis de compor a carteira.

Esse ativo livre de risco pode, inclusive, ser introduzido no Modelo (1.4), o que tornaria semidefinida positiva a matriz Q. Com essa alteração, todo o dinheiro disponível seria aplicado, exceto no caso em que $\lambda = 1$, situação para a qual há infinitas soluções ótimas, pois qualquer montante investido no ativo livre de risco seria ótimo, desde que nada fosse aplicado nos demais ativos. Naturalmente, a introdução de um ativo livre de risco altera a fronteira eficiente, fazendo com que esta toque o eixo vertical apresentado na Figura 1.1.

A analise das restrições de (1.4) revela que limitar o número de ativos na carteira nada mais é do que exigir que não mais que K componentes do vetor x sejam positivas. Logo, chamando de $nz(x) = \{i \mid x_i \neq 0\}$ o conjunto das componentes não nulas do vetor x, podemos substituir a restrição $\sum_{i=1}^{N} \delta_i \leq K$ pela exigência de que a cardinalidade de $nz(x)$ seja, no máximo, K, ou seja, que $Card(nz(x)) \leq K$.

Para manter as restrições de limite inferior, assim como as de positividade, basta exigir que as variávies cujos índices estam em *nz(x)* satisfaçam as restrições de limite inferior, permitindo que as demais sejam não negativas. Sendo assim, as 4 últimas restrições de (1.4) podem ser substituídas por

$$\begin{aligned} &Card(nz(x)) \le K \\ &x_i \ge l_i, \ \ i \in nz(x) \\ &x_i = 0, \ \ i \notin nz(x). \end{aligned}$$

Assim, podemos reescrever o problema (1.4) na forma

$$\begin{aligned} \min \quad & \lambda(\tfrac{1}{2} x^T Q x) - (1-\lambda)\mu^T x \\ s.a \quad & \sum_{i=1}^{N} x_i \le 1 \\ & Card(nz(x)) \le K \\ & x_i \ge l_i, \ \ i \in nz(x) \\ & x_i = 0, \ \ i \notin nz(x). \end{aligned} \tag{1.5}$$

Como essa mudança pode ser generalizada para qualquer problema de programçao quadrática, de uma forma geral, escrevemos nosso problema de programação quadrática mista 0-1 no seguinte formato:

$$\begin{aligned} \min \quad & \tfrac{1}{2} x^T Q x + c^T x \\ s.a. \quad & Ax \le b \\ & Card(nz(x)) \le K \\ & x_i \ge l_i, \ \ i \in nz(x) \\ & x_i = 0, \ \ i \notin nz(x), \end{aligned} \tag{1.6}$$

onde $Q \in \Re^{N \times N}$ é simétrica e definida positiva, $c \in \Re^N$, $A \in \Re^{m \times N}$, $b \in \Re^m$, $l \in \Re^N$ é o vetor de limites inferiores não negativos de *x*, *K* é um inteiro positivo, o conjunto *nz(x)* é definido por $nz(x) = \{i \mid x_i \neq 0\}$ e *Card(C)* indica a cardinalidade do conjunto *C*.

Além do Problema (1.5), outro tipo de problema que pode ser escrito na forma (1.6) é obtido quando queremos construir um *portfolio* limitado a um máximo de *K* ações que, ao mesmo tempo,

- se equipare, o tanto quanto possível, a um *portfolio* alvo composto por *N* ações (*N* > *K*);
- minimize os custos transacionais;
- limite a mudança total de investimentos entre setores da economia.

Suponha que disponhamos de um *portfolio* atual, com frações de investimento em seus *N* ativos dadas pelo vetor x^0. Assumindo a adoção de modelo simétrico para compras e vendas, podemos minimizar os custos associados às alterações na carteira incluindo na função objetivo o termo quadrático

$$\sum_{i=1}^{N} c_i (x_i - x_i^0)^2,$$

onde $c_i > 0$ é o coeficiente de impacto relativo à movimentação do ativo *i*.

Para limitar o aumento ou a redução dos investimentos que fazem parte de um setor do mercado de ações, exigimos que

$$\left| \sum_{i \in S_j} \left(x_i - x_i^B \right) \right| \leq \varepsilon_j, \tag{1.7}$$

onde ε_j é a variação admissível para a soma das aplicações que pertencem ao conjunto S_j, que representa o setor *j* da economia, e x_i^B é fração alvo para o ativo *i*. Dentre possíveis setores da economia que poderiam compor S_j podemos citar os setores bancário, agropecuário e petrolífero.

Transformando a Restrição (1.7) em um par de desigualdades lineares, obtemos o seguinte problema

$$
\begin{aligned}
\min \quad & -\mu^T x + \frac{1}{2}\left(x - x^B\right)^T Q\left(x - x^B\right) + \sum_{i=1}^{N} c_i \left(x_i - x_i^0\right)^2 \\
s.a \quad & \sum_{i \in S_j} x_i \leq \sum_{i \in S_j} x_i^B + \varepsilon_j, && j = 1,\ldots,k \\
& \sum_{i \in S_j} x_i \geq \sum_{i \in S_j} x_i^B - \varepsilon_j, && j = 1,\ldots,k \\
& \sum_{i=1}^{N} x_i = 1 \\
& Card\,(nz(x)) \leq K \\
& x_i \geq l_i && i \in nz(x) \\
& x_i \geq 0 && i = 1,\ldots,N,
\end{aligned}
$$

onde o termo $\left(x - x^B\right)^T Q\left(x - x^B\right)$ representa o risco do *porifolio* almejado se afastar do *porifolio* alvo.

Claramente, o Problema (1.7) tem a forma (1.6), de modo que pode ser solucionado usando nossa metodologia.

CAPÍTULO 2

O MÉTODO DE LEMKE

2.1 INTRODUÇÃO

Em nosso trabalho, estamos interessados em encontrar a solução ótima exata de problemas de programação quadrática inteira mista dados na forma (1.6). Para tanto, usaremos um método do tipo *Branch-and-Bound* implícito, conforme será apresentado no Capítulo 3.

A cada passo do algoritmo *Branch-and-Bound*, precisamos resolver um problema relaxado, do qual excluímos as variáveis inteiras e as restições que as envolvem. Obtemos, assim, um problema na forma

$$\begin{aligned} \min \quad & \tfrac{1}{2} x^T Q x + c^T x \\ s.a \quad & Ax \le b \\ & x \ge 0. \end{aligned} \tag{2.1}$$

Dentre as diversas alternativas existentes para a resolução desse problema, optamos por usar o método de Lemke, pois ele é particularmente eficiente quando começamos de uma solução infactível próxima à solução ótima, situação comumente encontrada durante a aplicação do método *Branch-and-Bound*, como será visto no próximo capítulo.

2.2 O PROBLEMA DE COMPLEMENTARIDADE LINEAR

O método de Lemke, utilizado para otimizar os problemas relaxados associados aos nós da árvore gerada pelo algoritmo *Branch-and-Bound*, foi criado originalmente para resolver problemas de complementaridade linear (PCL), que são apresentados da seguinte forma:

- Dados $q \in \Re^n, M \in \Re^{n \times n}$, achar $z \in \Re^n$ e $w \in \Re^n$ tais que:

$$\begin{aligned} & w = Mz + q \\ & z \geq 0, w \geq 0 \\ & z^T w = 0. \end{aligned} \tag{2.2}$$

Esse problema é conhecido como PCL(*q,M*). Um vetor *z* é dito *factível* se satisfaz as duas primeiras restrições de (2.2), e é dito *complementar* se satisfaz a última equação. Todo vetor *z* que é factível e complementar é chamado de ponto de equilíbrio do PCL(*q,M*). Note que, devido às restrições de não negatividade, *z* e *w* são complementares se e somente se

$$z_i w_i = 0, \forall i = 1,...,n .$$

Chamamos o par z_i e w_i de *par complementar*.

Voltando ao Subproblema (2.1), observamos que, se *Q* é uma matriz simétrica e definida positiva, as condições de Karush-Kuhn-Tucker são necessárias e suficientes para a caracterização de um mínimo global do problema. Tais condições KKT são definidas por

$$
\begin{aligned}
& c + Qx + A^T g - r = 0 \\
& g^T (b - Ax) = 0 \\
& r^T x = 0 \\
& x,\ g,\ r \geq 0,
\end{aligned}
$$

onde $g \in \Re^m$ *e* $r \in \Re^N$ são denominadas *variáveis duais*. Com o uso de um vetor auxiliar $v \in \Re^m$, podemos reescrever as condições acima como

$$
\begin{aligned}
& r = c + Qx + A^T g \\
& v = b - Ax \\
& r^T x = 0,\ v^T g = 0 \\
& x,\ g,\ r,\ v \geq 0.
\end{aligned}
\tag{2.3}
$$

Utilizando a representação

$$
q = \begin{bmatrix} c \\ b \end{bmatrix},\ M = \begin{bmatrix} Q & A^T \\ -A & 0 \end{bmatrix},\ z = \begin{bmatrix} x \\ g \end{bmatrix},\ w = \begin{bmatrix} r \\ v \end{bmatrix}, \tag{2.4}
$$

é possível reescrever (2.3) na forma

$$
\begin{aligned}
& w - Mz = q \\
& w^T z = 0 \\
& w,\ z \geq 0.
\end{aligned}
\tag{2.5}
$$

A primeira Equação de (2.5) pode ser apresentada simplesmente como

$$
\begin{bmatrix} -M & I_n \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix} = q, \tag{2.6}
$$

onde I_n é a matriz identidade de ordem $n = N + m$.

Podemos encontrar uma solução factível para (2.3) escolhendo n colunas linearmente independentes da matriz $\begin{bmatrix} I_n & -M \end{bmatrix}$ e formando uma base B tal que $B^{-1}q \geq 0$. Neste caso, apenas as variáveis correspondentes às colunas da base não são nulas. Sendo assim, podemos particionar o nosso vetor $\bar{x} = \begin{bmatrix} z \\ w \end{bmatrix}$ em dois outros vetores, $\bar{x}_B$ e $\bar{x}_{NB}$, sendo o primeiro composto pelas varíaveis não nulas (básicas) e o segundo pelas variáveis nulas (não básicas).

Desprezando as variáveis não básicas, podemos escrever (2.6) como $B\bar{x}_B = q$. Para atender as restrições de não negatividade de (2.3), exigimos que $\bar{x}_B = B^{-1}q \geq 0$. Além disso, para que as restrições de complementaridade sejam satisfeitas, z_i será básica se e somente se w_i não for básica, e vice-versa.

Seja, então, β o conjunto dos índices de z que estão na base. Neste caso, definimos a $i^{\text{ésima}}$ coluna da base complementar (aquela que satisfaz as condições de complementaridade) por

$$B_i = \begin{cases} -M_i, & se \quad i \in \beta, \\ e_i, & se \quad i \notin \beta, \end{cases}$$

onde e_i é a $i^{\text{ésima}}$ coluna da matriz identidade. Assim, cada coluna de B associada a uma componente de z é igual a uma coluna de M com o sinal trocado, enquanto as colunas de B associadas às variáveis de w são colunas da identidade. O objetivo do método de Lemke é achar uma base complementar B, tal que $B^{-1}q \geq 0$.

2.3 O MÉTODO DE LEMKE

Dados q, M e um vetor h positivo definido pelo usuário, o método de Lemke resolve o PCL (q,M) através de uma série de pivoteamentos.

Primeiramente, verificamos se $q \geq 0$, pois, nesse caso, temos a solução trivial $z = 0$ e $w = q$ como solução do problema. Caso isso não aconteça, precisamos definir uma base complementar inicial. As variáveis básicas, entretanto, não precisam satisfazer a restrição de não negatividade. A estratégia padrão, nesse caso, é fazer com que as componentes de z sejam não básicas e as de w sejam básicas, de modo que a base inicial seja a matriz identidade.

Em seguida, aumentamos o PCL(q,M) , substituindo (2.4) por

$$\begin{aligned} &w = q + hz_0 + Mz \\ &z^t w = 0 \\ &w \geq 0,\ z_0 \geq 0,\ z \geq 0. \end{aligned} \tag{2.7}$$

Essa alteração implica na ampliação de (2.6) para

$$\begin{bmatrix} -h & -M & I_n \end{bmatrix} \begin{bmatrix} z_0 \\ z \\ w \end{bmatrix} = q, \tag{2.8}$$

ou simplesmente $\overline{M}\overline{x} = q$, com $\overline{M} = \begin{bmatrix} -h & -M & I_n \end{bmatrix} \in \Re^{n\times(2n+1)}$ e $\overline{x} = \begin{bmatrix} z_0 & z^T & w^T \end{bmatrix}^T \in \Re^{2n+1}$.

A idéia implícita no aumento do PCL é a de que, ao introduzirmos no problema a coluna positiva h relativa a z_o e forçarmos tal variável a entrar na base, tornamos a nossa solução básica positiva.

Para mostrar que isso acontece, consideremos que a base inicial seja a trivial. Neste caso, queremos que $w \geq 0$, o que implica em $q + hz_0 \geq 0$, pois $z = 0$. Escolhendo o vetor h de forma adequada[1] e definindo

$$z_0 = \max(-q_i / h_i),$$

garantimos que a solução obtida após a transformação de z_0 em variável básica seja necessariamente positiva. Nesse caso, a variável que é forçada a sair da base para a inclusão de z_0 é exatamente aquela definida por argmax$\{-q_i/h_i\}$.

A introdução de z_0 faz com que um par (z_i, w_i) deixe de ser complementar, pois as duas variáveis que o compõem passam a ser não básicas. Para corrigir esse problema, o método de Lemke escolhe, a cada iteração, a variável complementar àquela que saiu da base na iteração anterior para entrar na base da iteração atual.

A variável que entra na base pode ter seu valor aumentado, até que uma das variáveis básicas se torne zero. A primeira variável básica a atingir o valor zero (denominada *variável de bloqueio*) deixa a base. O processo de determinação da variável que sai da base é análogo ao teste da razão, utilizado na Programação Linear.

Repetimos o processo de substituição de variáveis na base até que z_0 se torne a variável de bloqueio, ou até que não consigamos achar uma variável de bloqueio. No primeiro caso, z_0 é retirada da base e, assim, obtemos uma solução que é ao mesmo tempo factível e complementar, de modo que dispomos da solução ótima. No segundo caso, o nosso PCL original (2.5) (e, conseqüentemente, nosso problema de otimização quadrática) é infactível .

Na ausência de degenerescência e de uma região factível vazia, podemos garantir que o método de Lemke convirgirá para uma solução do problema (2.1) caso ao menos uma das condições abaixo seja satisfeita[2]:

---

[1] Quando se usa a base trivial, uma opção simples consiste em definir $h = [1\ 1\ ...\ 1]^T$.

[2] A demonstração desse resultado pode ser encontrada em Bazaraa *et al.*[2].

- A matriz Q seja simétrica e semidefinida positiva e $c = [0\ 0\ ...\ 0]^T$.
- A matriz Q seja simétrica definida positiva.
- A matriz Q seja simétrica, possua todos os elementos não negativos e sua diagonal seja composta apenas por elementos positivos.

Em nosso trabalho, supomos que a matriz Q é simétrica e definida positiva, de modo que o método de Lemke sempre convergirá para o minimizador do problema quadrático. Além disso, é fácil notar que o algoritmo de Lemke termina em um número finito de passos, pois o número de bases complementares é finito. Entretanto, a seqüência de pivoteamentos depende diretamente da escolha do vetor h, sendo importante a escolha de um vetor que torne o algoritmo mais eficiente.

Durante a execução do algoritmo *Branch-and-Bound*, aplicamos o método de Lemke para resolver uma série de problemas quadráticos semelhantes. Para acelerar o algoritmo, aproveitamos a solução obtida para um problema como ponto inicial na resolução de outro problema. Neste caso, não é possível utilizar como base inicial, a base ótima de um problema anterior, como exposto acima, sendo necessário definir também um novo vetor h apropriado. Uma descrição de como introduzir essa alteração no método de Lemke é apresentada no Capítulo 4, no qual abordamos nossa implementação computacional.

2.4 ALGORITMO

A seguir, apresentamos o método de Lemke na forma algoritmica, partindo da solução inicial trivial. Supomos que sejam fornecidos os valores de q, h, e M. No algoritmo abaixo, os índices de 0 a n correspondem ao vetor z, e os índices de $n + 1$ a $2n$ dizem respeito ao vetor w. Ao longo do algoritmo, usaremos a notação $\overline{M}(:, I_B)$ para representar a submatriz de $\overline{M}$ que contém apenas as colunas relacionadas aos índices do conjunto I_B.

1. **Se** $q \geq 0$, (a solução trivial $z = 0$ é ótima)

   1.1. $z \leftarrow 0,\ w \leftarrow q$.

   1.2. Terminar o algoritmo.

2. **Fim Se**

3. Definir os vetores de índices:

   $I_B \leftarrow \{n+1, ..., 2n\},\ I_N \leftarrow \{0, ..., n\}$

4. Introduzir a coluna de z_0 na base:

   $i \leftarrow \arg\max\{-q_j / h_j\}$.

   $I_B \leftarrow (I_B \setminus \{i\}) \cup \{0\},\ I_N \leftarrow (I_N \setminus \{0\}) \cup \{i\}$

   $B \leftarrow \overline{M}(:, I_B)$

5. **Enquanto** $0 \in I_B$ (a variável z_0 é básica),

   5.1. Forçar a entrada na base da variável complementar àquela que saiu na iteração anterior:

   $$j \leftarrow i + n - 2n\lfloor i/(n+1) \rfloor.$$

   5.2. Determinar a variável que sai da base pelo teste da razão:

   $$i \leftarrow \arg\min\left\{\left(\mathrm{B}^{-1}q\right)_k \Big/ \left(\mathrm{B}^{-1}\overline{m}_j\right)_k,\ \left(\mathrm{B}^{-1}\overline{m}_j\right)_k > 0\right\},$$

   onde $\overline{m}_j$ é a j$^{\text{ésima}}$ coluna da matriz $\overline{M}$.

   5.3. Se i estiver definido,

      5.3.1. Atualizar os vetores de índices e a base:

      $I_B \leftarrow (I_B \setminus \{i\}) \cup \{j\},\ I_N \leftarrow (I_N \setminus \{j\}) \cup \{i\}$

      $B \leftarrow \overline{M}(:, I_B)$

   5.4. Senão,

      5.4.1. O problema é infactível.

      5.4.2. Terminar o algoritmo.

6. **Fim Enquanto.**

7. $\bar{x} \leftarrow B^{-1}q$.

2.5 EXEMPLO NUMÉRICO

Para facilitar a compreensão do método de Lemke, apresentamos abaixo um exemplo de aplicação do mesmo a um problema simples na forma (2.1), usando como dados de entrada os vetores e matrizes fornecidos abaixo.

$$A = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \; b = \begin{bmatrix} 1 \end{bmatrix}, \; c = \begin{bmatrix} -1{,}8 \\ -2{,}5 \end{bmatrix}, \; h = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Vamos resolver o problema aumentado na forma (2.7), cuja dimensão é 7, partindo da base inicial trivial. As iterações serão apresentadas com o auxílio de tablôs. Nesses tablôs, as variáveis básicas são aquelas indicadas na primeira coluna, enquanto os valores dessas variáveis são mostrados na última coluna.

Aplicando o Algoritmo de Lemke ao problema, temos

Iteração 0.

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
w_1	–1	–2	–1	–1	1	0	0	–1,8
w_2	–1	–1	–2	–1	0	1	0	–2,5
w_3	–1	1	1	0	0	0	1	1

- Variável que sai da base: w_2
- Variável que entra na base: z_0
- Novo tablô:

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
w_1	0	–1	1	0	1	–1	0	0,7
z_0	1	1	2	1	0	–1	0	2,5
w_3	0	2	3	1	0	–1	1	3,5

Iteração 1.

- Variável que sai da base: w_1
- Variável que entra na base: z_2
- Novo tablô:

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
z_2	0	–1	1	0	1	–1	0	0,7
z_0	1	3	0	1	–2	1	0	1,5
w_3	0	5	0	1	–3	2	1	1,4

Iteração 2.

- Variável que sai da base: w_3
- Variável que entra na base: z_1
- Novo tablô:

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
z_2	0	0	1	0,2	0,4	–0,6	0,2	0,98
z_0	1	0	0	0,4	–0,2	–0,2	–0,6	0,26
z_1	0	1	0	0,2	–0,6	0,4	0,2	0,28

Iteração 3.

- Variável que sai da base: z_0
- Variável que entra na base: z_3
- Novo tablô:

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
z_2	–0,5	0	1	0	0,5	–0,5	0,5	0,85
z_3	2,5	0	0	1	–0,5	–0,5	–1,5	0,65
z_1	–0,5	1	0	0	–0,5	0,5	0,5	0,15

Iteração 4.

- Como z_0 sai da base, a solução atual é ótima para o nosso problema. Sabendo que as duas primeiras componentes de z correspondem ao vetor x, concluímos que $x_1 = 0,15$ e $x_2 = 0,85$.

2.6 RESOLVENDO AS ITERAÇÕES DO MÉTODO DE LEMKE

Observamos que o método de Lemke possui uma grande semelhança com o método Simplex, especialmente no que se refere ao uso de pivoteamentos e do teste da razão para determinar a variável que sai da base. Cabe notar, também, que, como na fase 1 do Simplex, tornamos factível a solução inicial incluindo uma ou mais variáveis artificiais no problema. Porém, diferentemente do Simplex, o método de Lemke não lida com a função objetivo e, conseqüentemente, com os custos associados a cada variável. Sendo assim, o critério de otimalidade e a estratégia usada para determinar a variável que entra na base são diferentes dos empregados na resolução do Simplex.

Dadas as semelhanças entre os algoritmos, podemos adaptar ao método de Lemke várias propostas desenvolvidas para o Simplex. Como os passos computacionalmente mais caros do método estão relacionados à resolução de sistemas lineares envolvendo a matriz B, consideramos duas possibilidades de implementação do método de Lemke. A primeira envolve a decomposição LU da base sempre que uma coluna é substituída por outra. A segunda proposta consiste em efetuar a decomposição LU no início do método, atualizando as matrizes L e U a cada iteração. As duas alternativas foram implementadas e testadas, conforme se verá adiante.

CAPÍTULO 3

O MÉTODO *BRANCH-AND-BOUND* IMPLÍCITO

3.1 INTRODUÇÃO

Uma vez que o Problema (1.6) é equivalente a um problema de programação quadrática inteira mista, a maneira usual de encontrar sua solução exata consiste em utilizar variantes do algoritmo *Branch-and-Bound*. Dessa forma, os métodos que investigamos para a resolução desse tipo de problema foram:

- A aplicação direta do método *Branch-and-Bound*, combinado com um algoritmo para resolver problemas convexos de programação quadrática. Essa estratégia, apesar de ser a mais intuitiva, é mais cara, em termos de gasto computacional, que os métodos citados abaixo.
- O uso do algoritmo *Branch-and-Cut*, que inclui no problema restrições baseadas em métodos de planos de corte, sempre que possível. Seguindo essa linha, Bienstock [4] investigou quatro estratégias diferentes de geração de planos de corte para o problema: *cortes arredondados inteiro-mistos, cortes de mochila, cortes de intersecção e cortes disjuntos*. Dentre elas, a última se mostrou mais eficiente devido ao fato da separação ser feita de maneira direta.

- O uso do algoritmo *Branch-and-Bound* implícito, combinado com o *método de Lemke,* para a resolução dos subproblemas quadráticos, além de técnicas de reformulação e eliminação de variáveis.

Dentre as estratégias descritas, a que mais nos pareceu promissora foi a última, uma vez que o método de Lemke pode ser facilmente adaptado à resolução dos problemas gerados a cada iteração do *Branch-and-Bound*, problemas estes que possuem, geralmente, um número muito pequeno de restrições e para os quais dispomos de uma solução inicial quase factível.

3.2 O MÉTODO *BRANCH-AND-BOUND* PARA PROBLEMAS 0-1

O *Branch-and-Bound* é um algoritmo baseado no processo de divisão e conquista, que consiste em quebrar um problema difícil (em geral, NP-completo) em vários problemas menores, de fácil resolução, para posteriormente juntar as informações obtidas para resolver o problema original. Dessa forma, é feita a enumeração sistemática das possíveis soluções, procurando sempre eliminar, ao longo do caminho, grupos de soluções menos proveitosas. Em geral, esse algoritmo é muito utilizado para achar a solução ótima de problemas de otimização que envolvem variáveis inteiras.

Neste trabalho, estamos interessados em resolver problemas de programação quadrática mista do tipo (1.4), em que algumas variáveis são binárias, ou seja, só podem assumir valores 0 ou 1. Para resolver esse tipo de problema usando o *Branch-and-Bound*, primeiramente resolvemos a versão relaxada do problema, o que no nosso caso corresponderia a resolver o problema no qual todas as variáveis binárias são consideradas contínuas no intervalo [0,1]. Feito isso, verificamos se, na solução obtida, todas as variáveis binárias valem 0 ou 1. Se isso ocorre, já resolvemos o problema. Caso contrário, como ainda não encontramos uma solução viável para o problema original, guardamos uma solução vazia como a melhor soluçao inteira. Neste caso,

definindo ζ^* como o valor da função objetivo desta melhor solução, tomamos $\zeta^* = \infty$.

Em seguida, escolhemos uma das variáveis binárias que ora possue valor fracionário e a forçamos a assumir explicitamente cada um dos dois valores admissíveis (0 ou 1). Geramos, assim, dois subproblemas que precisam ser resolvidos. A essa subdivisão do problema damos o nome de ramificação. Nesse contexto, existem 2 tipos diferentes de ramificação: a para baixo e a para cima. Na ramificação para baixo, a variável escolhida assume o valor 0, ao passo que na ramificação para cima ela assume o valor 1.

É fácil observar que cada ramificação dá origem a outras ramificações, até que todas as variáveis inteiras da solução obtida possuam valor 0 ou 1. Assim, a estrutura do problema sugere a criação de uma árvore binária para poder representar todas as possíveis ramificações a serem investigadas.

Nessa árvore, cada subproblema gerado por uma ramificação é chamado de *nó*. O primeiro problema relaxado a ser resolvido, é conhecido como *nó raiz*. Um nó cuja solução tenha as variáveis binárias valendo 0 ou 1 é chamado de *folha*, ao passo que um nó em que algumas das variáveis binárias estejam em (0,1) é chamado de *ramo*. Quando um nó *i* gera um nó *j* via ramificação, dizemos que o nó *i* é o *pai* de *j*, de modo que este último é um nó *filho* de *i*. Um nó que ainda não foi resolvido é chamado de nó *pendente*. O número máximo de elementos da árvore é igual a $2^{n+1} - 1$, onde *n* é o número de variáveis binárias do problema.

Depois de resolver o problema associado a uma ramificação, verificamos se a solução é factível, no sentido das variáveis binárias do problema assumirem apenas valores 0-1. Se isso ocorre, comparamos ζ, o valor da função objetivo, com ζ^*, o valor de ζ associado à melhor solução inteira encontrada até o momento. Se $\zeta < \zeta^*$, atualizamos ζ^* e deixamos de resolver todas as ramificações pendentes cuja solução seja pior que aquela que acabamos de encontrar. Por outro lado, se a solução ainda incluir variáveis binárias com valores não inteiros, escolhemos uma nova variável para ramificar, gerando dois novos problemas.

Salvo quando aparece uma folha, a cada iteração do método, temos que armazenar 2 novos nós e decidir qual nó resolver dentre os possíveis pendentes. O número de nós pendentes só diminui quando achamos uma folha, pois, neste caso, além de não ramificarmos, eliminamos os nós pendentes com valor de ζ maior que o da solução recém encontrada. O algoritmo acaba

quando não existem mais nós pendentes para investigarmos.

Para facilitar o entendimento do leitor, apresentamos abaixo o algorimo *Branch-and-Bound* para problemas de programação quadrática mista 0-1.

1. Usando o método de Lemke, resolver o problema relaxado (assumindo que as variáveis binárias são contínuas no intervalo [0,1]).
2. **Se**, na solução obtida, todas as variáveis binárias valerem 0 ou 1,
   2.1. Terminar o algoritmo (estamos no ótimo).
3. **Fim Se**.
4. Escolher uma variável binária não inteira para ramificar.
5. Criar uma lista de nós pendentes, *nosp*, contendo os 2 problemas gerados pela ramificação.
6. $x^* \leftarrow [\,]$ (vetor indefinido). $\zeta^* \leftarrow \infty$.
7. **Enquanto** $nosp \neq \varnothing$,
   7.1. Retirar um subproblema de *nosp*.
   7.2. Usando o método de Lemke, resolver este problema, obtendo x e ζ ou a indicação de que ele é infactível.
   7.3. **Se** x possuir variáveis binárias com valores fracionários,
      7.3.1. Escolher uma variável para ramificar.
      7.3.2. Incluir em *nosp* os 2 subproblemas a serem resolvidos.
   7.4. **Senão, Se** as variáveis binárias de x forem inteiras e $\zeta < \zeta^*$,
      7.4.1. $x^* \leftarrow x$. $\zeta^* \leftarrow \zeta$.
      7.4.2. **Para** cada nó i em *nosp*,
         7.4.2.1. **Se** $\zeta_i^p \geq \zeta^*$, onde ζ_i^p é o valor da função objetivo do pai do nó i,
            7.4.2.1.1. Eliminar o nó i de *nosp*.
         7.4.2.2. **Fim Se.**
      7.4.3. **Fim Para**.
   7.5. **Fim Se**.
8. **Fim Enquanto**.
9. A solução do problema é x^*, com função objetivo igual a ζ^*.

Para obter maiores informações sobre o algoritmo *Branch-and-Bound*, o leitor pode consultar, por exemplo, Wolsey [16].

3.3 O MÉTODO *BRANCH-AND-BOUND* IMPLÍCITO

A principal diferença entre o Problema (1.5) ou (1.6) e um problema de programação quadrática 0-1 convencional é que as variáveis binárias estão implícitas, ou seja, não aparecem, mas são necessárias para que tenhamos no máximo K variáveis positivas, ou seja, para que

$$Card(nz(x)) \leq K \,. \tag{3.1}$$

Essa forma implícita de tratar as variáveis binárias pode ser estendida ao método *Branch-and-Bound*. Neste caso, eliminamos do problema relaxado (o nó raiz da árvore binária) a Restrição (3.1) e os limites

$$\begin{aligned} x_i &\geq l_i, \;\; i \in nz(x) \\ x_i &= 0, \;\; i \notin nz(x), \end{aligned} \tag{3.2}$$

adotando apenas a restrição de não negatividade $x \geq 0$.

Obtemos, assim, um problema clássico de programação quadrática, na forma (2.1). A única deferença entre este problema e o correspondente ao nó raiz do *Branch-and-Bound* usual é a inexistência da restrição $\sum_{i=1}^{N} \delta_i \leq K$, presente na formulação (1.4), mas não em (1.5). Ainda assim, a relaxação das restrições de integralidade das variáveis binárias, adotada na seção anterior, também permite que a cardinalidade de $nz(x)$ seja superior a K e que as componentes

estritamente positivas do vetor x assumam valores abaixo de seus limites inferiores, de modo que há grande similaridade entre os problemas abordados pelo *Branch-and-Bound* usual e a versão implícita.

No caso provável da solução do nó raiz da árvore não satisfazer as restrições (3.1) ou (3.2), é necessário ramificar uma variável qualquer. É justamente na manipulação das ramificações que as restrições ignoradas são levadas em conta, justificando o caráter implícito do método *Branch-and-Bound*. A idéia é fazer com que, a cada ramificação, as variáveis estejam mais próximas de obedecer às restrições de cardinalidade e limite inferior.

Vejamos como ramificar uma variável x_s:

- Habitualmente, a ramificação para baixo corresponde à atribuição do valor 0 à variável binária associada a x_s. Naturalmente, isso também nos obriga a adotar $x_s = 0$. No método implícito, essa ramificação é obtida através da eliminação direta de x_s, o que provoca a geração de um novo subproblema com uma variável a menos. Esse subproblema é resolvido através do método de Lemke.

- No problema financeiro usual, a ramificação para cima está relacionada à inclusão forçada de um ativo na carteira. No *Branch-and-Bound* para problemas 0-1, isso equivale a atribuir o valor 1 à variável binária associada à variável x_s, ao passo que, no *Branch-and-Bound* implícito, tal exigência se resume à inclusão da restrição $x_s \geq l_s$. Esse tipo de ramificação traz conseqüências não só à restrição de limite inferior, mas também à restrição de cardinalidade, que pode passar a ser infactível. No método implícito, essa ramificação é feita apenas quando x_s é estritamente positiva e está abaixo do seu limite inferior l_s. Neste caso, incluímos no problema a restrição $x_s \geq l_s$ e o resolvemos pelo método de Lemke. Para evitar que a Restrição (3.1) se torne infactível, a ramificação para cima não é feita se o número de variáveis ramificadas para cima for superior ou igual a K.

Como veremos a seguir, os principais conceitos do algoritmo *Branch-and-Bound* são mantidos em sua versão implícita, que pouco difere daquela apresentada na Seção 3.2.

3.4 ALGORITMO E EXEMPLO

Nesta seção, para facilitar a compreensão da versão implícita do *Branch-and-Bound,* apresentamos o passos que compõem o algoritmo, além de um exemplo numérico.

3.4.1 Algoritmo

1. Usando o método de Lemke, resolver o problema relaxado, retirando as Restrições (3.1) e (3.2).
2. **Se**, na solução obtida, não tivermos mais que K variáveis não nulas e se estas satisfizerem as restrições de limite inferior,
   2.1. Terminar o algoritmo (estamos no ótimo).
3. **Fim Se**.
4. Escolher uma variável binária não inteira para ramificar.
5. Criar uma lista de nós pendentes, *nosp*, contendo os 2 problemas gerados pela ramificação.
6. $x^* \leftarrow [\,]$ (vetor indefinido). $\zeta^* \leftarrow \infty$.
7. **Enquanto** $nosp \neq \varnothing$,
   7.1. Retirar um subproblema de *nosp*.
   7.2. Usando o método de Lemke, resolver este problema, obtendo x e ζ ou a indicação de que ele é infactível.
   7.3. **Se** x possuir componentes não nulas menores que os limites inferiores correspondentes ou se o número de componentes não nulas for maior que K,
      7.3.1. Escolher uma variável para ramificar.
      7.3.2. Incluir em *nosp* o problema relacionado à ramificação para baixo.
      7.3.3. **Se** a ramificação para cima não violar a Restrição (3.1),

7.3.3.1. Incluir a ramificação para cima em *nosp*.

7.3.4. **Fim Se**.

7.4. **Senão, Se** $\zeta < \zeta^*$,

7.4.1. $x^* \leftarrow x$. $\zeta^* \leftarrow \zeta$.

7.4.2. **Para** cada nó *i* de *nosp*,

7.4.2.1. **Se** $\zeta_i^p \geq \zeta^*$, onde ζ_i^p é o valor da função objetivo do pai do nó *i*,

7.4.2.1.1. Eliminar o nó *i* de *nosp*.

7.4.2.2. **Fim Se.**

7.4.3. **Fim Para**.

7.5. **Fim Se**.

8. **Fim Enquanto**.
9. A solução do problema é x^*, com função objetivo igual a ζ^*.

3.4.2 Exemplo

Consideremos o Problema (1.5) com $\lambda = 0{,}5$ e $K = 2$. Tomemos como dados iniciais

$$Q = \begin{bmatrix} 2.2 & 2 & 2 \\ 2 & 2.5 & 2 \\ 2 & 2 & 2.6 \end{bmatrix},\ \mu = \begin{bmatrix} 1{,}87 \\ 2{,}15 \\ 2{,}6 \end{bmatrix},\ l = \begin{bmatrix} 0{,}3 \\ 0{,}5 \\ 0{,}85 \end{bmatrix}.$$

Ao longo do algoritmo, representaremos cada subproblema da lista de nós pendentes (*nosp*) utilizando o vetor *ram*, cuja $i^{\text{ésima}}$ coordenada é definida conforme indicado abaixo:

- *ram*(i) = –1, se a variável x_i ainda não tiver sido ramificada;
- *ram*(i) = 0, se a variável x_i tiver sido ramificada para baixo;
- *ram*(i) = 1, se a variável x_i tiver sido ramificada para cima.

Mostramos abaixo, passo a passo, a aplicação do algoritmo.

- Passo 1: Resolvendo o nó raiz, obtemos $\zeta = -1{,}1167$ e $x = [0 \quad 0{,}364 \quad 0{,}62]^T$.
- Passo 2: Como as variáveis 2 e 3 estão abaixo do limite inferior e acima de zero, não estamos no ótimo
- Passo 4: Escolhemos x_3 para ramificar, apesar da escolha da segunda variável também ser plausível. Heurísticas para a seleção da variável a ser ramificada em caso de empate serão apresentadas na Seção 4.2.
- Passo 5: Entram na lista de nós pendentes as ramificações para cima e para baixo de x_3. Assim, *nosp* = { $[-1 \quad -1 \quad 0]$, $[-1 \quad -1 \quad 1]$ }.
- Passo 7:
  - Iteração 1:

    Passo 7.1: Retiramos o subproblema $[-1 \quad -1 \quad 0]$ de *nosp*. Logo, *nosp* = { $[-1 \quad -1 \quad 1]$ }.
    Heurísticas de como escolher o nó de da lista de pendentes serão fornecidas na Seção 4.3

    Passo 7.2: Resolvendo o subproblema, obtemos $\zeta = -0{,}9432$ e $x = [0{,}25 \quad 0{,}66 \quad 0]^T$.

    Passo 7.3: A componente 1 do vetor *x* está abaixo de seu limite inferior.

      Passo 7.3.1: Escolhemos x_1 para ramificar, pois esta é a única variável não nula abaixo de seu limite inferior

      Passo 7.3.2: Incluímos em *nosp* o nó correspondente à ramificação para baixo de x_1. Logo, *nosp* = { $[-1 \quad -1 \quad 1]$, $[0 \quad -1 \quad 0]$ }.

      Passo 7.3.3:A ramificação para cima de x_1 não viola (3.1). Logo, também a incluímos em *nosp*. Assim, *nosp* = { $[-1 \quad -1 \quad 1]$, $[0 \quad -1 \quad 0]$, $[1 \quad -1 \quad 0]$ }.

  - Iteração 2:

    Passo 7.1: Retiramos o subproblema $[-1 \quad -1 \quad 1]$ de *nosp*. Logo, *nosp* = { $[0 \quad -1 \quad 0]$, $[1 \quad -1 \quad 0]$ }.

Passo 7.2: Resolvendo o subproblema, obtemos $\zeta = -1{,}0891$ e $x = [0 \quad 0{,}015 \quad 0{,}85]^T$.

Passo 7.3: A componente 2 do vetor *x* está abaixo de seu limite inferior.

Passo 7.3.1: Escolhemos x_2 para ramificar, pois esta é a única variável não nula abaixo de seu limite inferior.

Passo 7.3.2: Incluímos em *nosp* o nó correspondente à ramificação para baixo de x_2. Logo, *nosp* = { $[0 \quad -1 \quad 0]$, $[1 \quad -1 \quad 0]$, $[-1 \quad 0 \quad 1]$}.

Passo 7.3.3: A ramificação para cima de x_2 viola (3.1). Logo, não a incluímos em *nosp*.

- Iteração 3:

Passo 7.1: Retiramos o subproblema $[-1 \quad 0 \quad 1]$ de *nosp*. Logo, *nosp* = { $[0 \quad -1 \quad 0]$, $[1 \quad -1 \quad 0]$}.

Passo 7.2: Resolvendo o subproblema, obtemos $\zeta = -1{,}0563$ e $x = [0{,}0773 \quad 0 \quad 0{,}85]^T$.

Passo 7.3: A componente 1 do vetor *x* está abaixo de seu limite inferior.

Passo 7.3.1: Escolhemos x_1 para ramificar, pois é a única variável não nula abaixo de seu limite inferior

Passo 7.3.2: Incluímos em *nosp* o nó correspondente à ramificação para baixo de x_2. Logo, , *nosp* = { $[0 \quad -1 \quad 0]$, $[1 \quad -1 \quad 0]$, $[0 \quad 0 \quad 1]$}.

Passo 7.3.3:A ramificação para cima de x_1 viola (3.1). Logo, o nó correspondente não é incluído em *nosp*.

- Iteração 4:

Passo 7.1 Retiramos o subproblema $[0 \quad 0 \quad 1]$ de *nosp*. Logo, *nosp* = { $[0 \quad -1 \quad 0]$, $[1 \quad -1 \quad 0]$}.

Passo 7.2: Resolvendo o subproblema, obtemos $\zeta = -1{,}053$ e $x = [0 \quad 0 \quad 0{,}9]^T$.

Passo 7.3: Como *x* não possui componente não nula menor que os seus limites inferiores e não viola a restrição de cardinalidade, encontramos uma solução inteiro-mista factível.

Passo 7.4: Observamos que $\zeta < \zeta^*$, pois esta é a primeira solução factível encontrada.

Passo 7.4.1: $x^* \leftarrow [0 \quad 0 \quad 0{,}9]^T$. $\zeta^* \leftarrow -1{,}053$.

Passo 7.4.2: Como $\zeta^P \geq \zeta^*$ para os nós da lista de nós pendentes, estes são eliminados e *nosp* = ∅.

- Passo 9: A solução é $\zeta^* = -1{,}053$ e $x^* = [0 \quad 0 \quad 0{,}9]^T$.

A Figura 3.1 mostra a árvore explorada durante o exemplo.

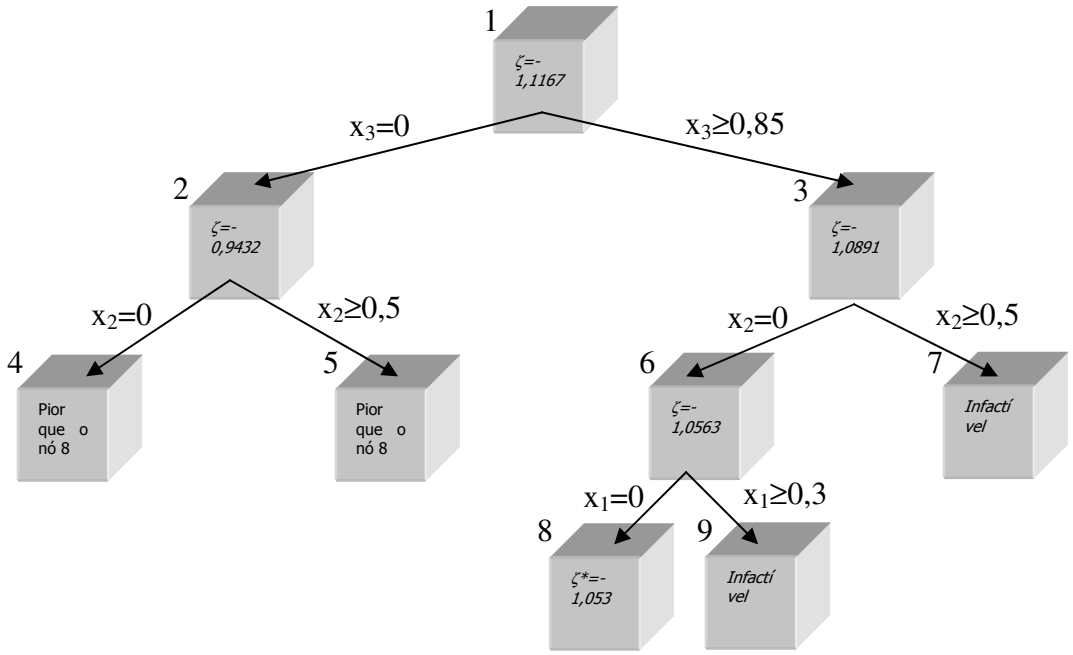


Figura 3.1 – Árvore binária gerada ao aplicarmos o algoritmo *Branch-and-Bound* ao exemplo.

3.5 RAMIFICANDO PARA BAIXO

Como visto na Seção 3.3, quando ramificamos para baixo uma variável x_s, ou seja, fixamos $x_s = 0$, eliminamos todos os dados a ela associados. Para submeter o problema assim gerado ao método de Lemke, é preciso definir uma nova base inicial. Naturalmente, não desejamos utilizar a base trivial, ou seja, a matriz identidade, pois isso aumentaria o tempo de execução do algoritmo. Felizmente, a base ótima do nó pai pode ser utilizada para a geração da base inicial, como veremos a seguir.

- Trabalhando com a decomposição LU da base.

Para descrever como obter uma nova base, assumamos que x_s é uma variável básica (caso ela seja não básica, podemos aplicar a mesma metolodologia para a variável complementar w_s, que será básica). Neste caso, devemos eliminar da matriz $\overline{M}$ do método de Lemke a linha e a coluna de correspondentes a x_s, assim como a coluna correspondente a w_s. Além disso, devemos excluir do vetor q a coordenada correspondente a c_s. Como conseqüência, a matriz B também perderá a linha e a coluna associadas a x_s.

Se formos capazes de obter a decomposição LU da nova matriz $\overline{B}$, essa matriz poderá ser usada como base inicial. Caso contrário, ou seja, se B for singular, teremos que começar o método de Lemke a partir de $\overline{B} = I_{n-1}$. Como opção, podemos aproveitar a decomposição LU já existente de B e atualizá-la, aproveitando a porção das matrizes L e U que antecede a linha excluída do problema.

- Atualizando a inversa da base

Apesar de usarmos a decomposição LU da base no algoritmo que implementamos,

também poderíamos obter a inversa da base inicial do método de Lemke a partir da inversa da base do nó pai aplicando operações elementares sobre as linhas desta matriz.

Para mostrar como isso é feito, consideremos que, ao aplicarmos o método de Lemke para o nó pai, tenhamos obtido o seguinte tablô ótimo:

$$\left[I \quad -M_B \mid \quad q_B \right].$$

onde $q_B = B^{-1}q$, $M_B = -B^{-1}N_b$ e $N_b \in \Re^{nxn}$ é a matriz das variáveis não básicas.

Para facilitar a nossa notação, supomos, sem perda de generalidade, que devemos eliminar a primeira linha e a primeira coluna de $B \in R^{nxn}$. Subdividindo, então, B e B^{-1} em termos da linha e da coluna a ser eliminada, obtemos

$$B = \begin{bmatrix} b_p & B_{lin}^T \\ B_{col} & \overline{B} \end{bmatrix}, B^{-1} = \begin{bmatrix} u_p & U_{lin}^T \\ U_{col} & \overline{U} \end{bmatrix},$$

onde $\overline{B}, \overline{U} \in \Re^{(n-1)x(n-1)}$; B_{col}, B_{lin}, U_{col}, $U_{lin} \in \Re^{n-1}$ e b_p e u_p são escalares. Sabemos que

$$B^{-1}B = \begin{bmatrix} u_p b_p + U_{lin}^T B_{col} & u_p B_{lin}^T + U_{lin}^T \overline{B} \\ b_p U_{col} + \overline{U} B_{col} & U_{col} B_{lin}^T + \overline{UB} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & I_{n-1} \end{bmatrix} = I_n .$$

Igualando à identidade o bloco inferior direito de $B^{-1}B$, obtemos:

$$\overline{UB} = I_{n-1} - U_{col} B_{lin}^T , \tag{3.1}$$

Observa-se que a matriz $U_{col} B_{lin}^T$ possui posto um. Aplicando uma série de operações elementares sobre as linhas da matriz $I_{n-1} - U_{col} B_{lin}^T$, podemos transformá-la novamente em I_{n-1}.

Chamemos de E a matriz que realiza tais operações. Dessa forma, temos $E\overline{UB} = I_{n-1}$. Logo, se $\overline{B}$ possuir inversa, esta será exatamente $E\overline{U}$.

Utilizando a mesma partição feita para a matriz das variáveis básicas, e considerando que a coluna correspondente à variável dual de x_s seja a primeira, podemos particionar a matriz das variáveis não básicas N_B da seguinte maneira:

$$N_B = \begin{bmatrix} n_p & N_{Blin}^T \\ N_{Bcol} & \overline{N}_B \end{bmatrix},$$

onde $\overline{N}_B \in \Re^{(n-1)x(n-1)}$; N_{Bcol}, $N_{Blin} \in \Re^{n-1}$ e n_p é um escalar. Dessa forma, podemos atualizar a matriz das variáveis não básicas M_B no tablô da seguinte maneira:

$$M_B = -B^{-1}N_B = -\begin{bmatrix} up & U_{lin}^T \\ U_{col} & \overline{U} \end{bmatrix}\begin{bmatrix} n_p & N_{Blin}^T \\ N_{Bcol} & \overline{N}_B \end{bmatrix} = -\begin{bmatrix} u_p n_p + U_{lin}^T N_{Bcol} & u_p N_{Blin}^T + U_{lin}^T \overline{N}_B \\ n_p U_{col} + \overline{U} N_{Bcol} & U_{col} N_{Blin}^T + \overline{U}\overline{N}_B \end{bmatrix} = \begin{bmatrix} m_p & M_{lin}^T \\ M_{col} & \overline{M}_B \end{bmatrix}$$

onde $\overline{M}_B \in R^{(n-1)x(n-1)}$; M_B, M_{col}, $M_{lin} \in \Re^{n-1}$ e m_p é um escalar. Novamente, igualando o bloco inferior direito das duas últimas matrizes, obtemos:

$$-\overline{UN}_B = \overline{M}_B + U_{col} N_{Blin}^T . \qquad (3.2)$$

Multiplicando os dois lados de (3.2) por E e lembrando que $E\overline{U}$ é a inversa de $\overline{B}$, obtemos a matriz atualizada

$$M_f = -\overline{B}^{-1}\overline{N}_B = E\left(\overline{M}_B + \overline{U}_{col} N_{Blin}^T\right).$$

Esse procedimento falha quando a nova base é singular. Neste caso, aplicamos o método de Lemke a partir da base trivial $\overline{B} = I_{n-1}$. Felizmente, ao longo dos testes computacionais realizados no Capítulo 5, tal fato pouco ocorreu.

O vetor do lado direito q_B, pode ser atualizado de forma semelhante, depois da eliminação do *s*-ésimo elemento do vetor *c*. Assumindo que, no tablô ótimo do nó pai, tínhamos $q_B = B^{-1}q$ com $q = \begin{bmatrix} c^T & b^T \end{bmatrix}^T$, e considerando que a primeira cordenada de q seja c_s, podemos reescrever q como $\begin{bmatrix} q_s & \overline{q}^T \end{bmatrix}^T$, onde $\overline{q} \in \Re^{n-1}$ e $q_s = c_s$. Neste caso,

$$q_B = \begin{bmatrix} \tilde{q}_s \\ \tilde{q}_B \end{bmatrix} = B^{-1}q = \begin{bmatrix} up & U_{lin}^T \\ U_{col} & \overline{U} \end{bmatrix} \begin{bmatrix} q_s \\ \overline{q} \end{bmatrix},$$

onde $\tilde{q}_B \in \Re^{n-1}$ e $\tilde{q}_s$ é um escalar. Utilizando o mesmo raciocínio empregado para obter M_f, chegamos ao seguinte vetor do lado direito atualizado

$$q_f = \overline{B}^{-1}\overline{q} = E\left(\tilde{q}_B - q_s U_{col} \right).$$

Agora, com as entradas do tablô atualizadas, podemos iniciar o método de Lemke para o nó filho a partir do tablô

$$\begin{bmatrix} I & -M_f \mid & q_f \end{bmatrix}.$$

3.5.1 Exemplo

Para ilustrar a ramificação para baixo, voltaremos ao problema utilizado, na Seção 2.5, para exemplificar a aplicação do método de Lemke. Para facilitar a leitura, reproduzimos abaixo o tablô ótimo do problema.

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
z_2	–0,5	0	1	0	0,5	–0,5	0,5	0,85
z_3	2,5	0	0	1	–0,5	–0,5	–1,5	0,65
z_1	–0,5	1	0	0	–0,5	0,5	0,5	0,15

Vamos supor que queiramos ramificar para baixo a variável z_1, retirando-a do problema. Neste caso, o novo tablô do método de Lemke é

VB	z_2	z_3	w_2	W_3	$B^{-1}q$
z_2	1	0	0	1	1
z_3	0	1	–1	–2	0,5

Como $B^{-1}q \geq 0$, já temos a solução do nó atual. Lembrando que as duas primeiras componentes de z correspondem ao vetor x, concluímos que $x_1 = 0$ e $x_2 = 1$.

3.6 RAMIFICANDO PARA CIMA

Quando ramificamos para cima uma variável x_s, devemos assegurar que $x_s \geq l_s$. Para tanto, basta fazermos uma mudança de variável, substituindo o vetor x pelo vetor y, onde

$$y_i = \begin{cases} x_i, & se \;\; i \neq s; \\ x_i - l_i & se \;\; i = s. \end{cases}$$

Dado que nosso conjunto de restrições era

$$\sum_{i \neq s} A_{.i} x_i + A_{.s} x_s \leq b \,,$$

onde $A_{.i}$ é a $i^{\text{ésima}}$ coluna de A, ao trocarmos x_s por $y_s + l_s$ obtemos

$$\sum_{i \neq s} A_{.i} y_i + A_{.s} y_s \leq b - l_s A_{.s} = \tilde{b} \,.$$

Obtemos, assim, um novo vetor do lado direito, $\tilde{b}$. Da mesma forma, a função objetivo deve ser alterada para

$$\frac{1}{2}(y + l_s e_s)^T Q(y + l_s e_s) + c^T (y + l_s e_s) \,.$$

Simplificando esta expressão, obtemos

$$\frac{1}{2} y^t Q y + \tilde{c}^T y + C_o \,,$$

onde $\tilde{c} = c + l_s Q_{.s}$ e $C_o = \frac{1}{2} {l_s}^2 Q_{s,s} + l_s c_s$ é uma constante. Uma vez que a constante C_o não influi no processo de otimização, podemos escrever nosso problema reformulado como

$$\begin{aligned} \min \quad & \tfrac{1}{2} y^T Q y + \tilde{c}^T y \\ s.a \quad & Ay \le \tilde{b} \\ & y \ge 0. \end{aligned}$$

Se B é a base do nosso problema, então o novo vetor q_B usado pelo método de Lemke será

$$q_B = B^{-1} q = B^{-1} \begin{bmatrix} \tilde{c} \\ \tilde{b} \end{bmatrix}.$$

Deve-se notar que apenas esse vetor do lado direito é alterado, mantendo-se a matriz do problema a ser resolvido pelo método de Lemke. Assim, se todas as componentes do novo vetor q_B são não negativas, a solução atual continua ótima. Como habitualmente isso não ocorre, é preciso determinar um vetor h adequado para garantir a não negatividade das variáveis básicas após a inclusão da coluna de z_0 na base. Um método eficiente para essa escolha de h será apresentado na Seção 4.4.

3.6.1 Exemplo

Voltemos ao exemplo da Seção 2.5 para, agora, ramificar para cima a variável z_2, colocando-a em seu limite inferior, que artificialmente foi escolhido como 0,9. Lembramos que o tablô final (reordenado) obtido para o problema original era

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
z_1	–0,5	1	0	0	–0,5	0,5	0,5	0,15
z_2	–0,5	0	1	0	0,5	–0,5	0,5	0,85
z_3	2,5	0	0	1	–0,5	–0,5	–1,5	0,65

Com a alteração do vetor $q_B = B^{-1}q$ e a escolha de um vetor h apropriado, que neste exemplo é o vetor trivial $[1\ 1\ ...\ 1]^T$, o novo tablô do método de Lemke passa a ser

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
z_1	–1	1	0	0	–0,5	0,5	0,5	0,15
z_2	–1	0	1	0	0,5	–0,5	0,5	–0,05
z_3	–1	0	0	1	–0,5	–0,5	–1,5	0,65

Observa-se que $z_2 < 0$, o que implica que a solução atual não é ótima. Assim, é preciso introduzir z_0 na base, de modo que o tablô final da iteração 0 é

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
z_1	0	1	–1	0	–1	1	0	0,2
z_0	1	0	–1	0	–0,5	0,5	–0,5	0,05
z_3	0	0	–1	1	–1	–0	–2	0,7

Iteração 1.

- Variável que sai da base: z_0
- Variável que entra na base: w_2
- Novo tablô:

VB	z_0	z_1	z_2	z_3	w_1	w_2	w_3	$B^{-1}q$
z_1	–2	1	1	0	0	0	1	0,1
w_2	2	0	–2	0	–1	1	–1	0,1
z_3	0	0	–1	1	–1	0	–2	0,7

Iteração 2.

- Como z_0 sai da base, a solução atual é ótima. Lembrando que as duas primeiras componentes de z correspondem ao vetor x, obtemos

  $x_1 = z_1 = 0,1;$
  $x_2 = l_2 + z_2 = 0,9 + 0 = 0,9.$

CAPÍTULO 4

PROCEDIMENTOS COMPUTACIONAIS

4.1 O PROJETO

Neste trabalho, implementamos um algoritmo para a resolução de problemas de otimização quadrática inteira mista utilizando o método *Branch-and-Bound* implícito combinado com o algoritmo de Lemke, conforme exposto anteriormente. Esse algoritmo foi desenvolvido de modo a resolver, especificamente, problemas de otimização de carteiras de investimentos.

Para tornar nosso algoritmo o mais eficiente possível, de modo que ele pudesse concorrer com outras bibliotecas destinadas à solução desse tipo de problema, incorporamos em nosso programa

- O aproveitamento da base ótima do nó pai, ou de um nó vizinho, como base inicial para o algoritmo de Lemke, quando este é aplicado à otimização de um nó da árvore do método *Branch-and-Bound*.
- O emprego de um procedimento eficiente para a determinação do vetor de cobertura *h* do algoritmo de Lemke, de modo que fosse possível utilizar qulaquer solução inicial, sem a necessidade de partir da base inicial trivial.

- O uso da decomposição LU da base na resolução dos sistemas lineares relacionados ao método de Lemke. Além disso, para reduzir o esforço computacional associado à resolução desses sistemas, também permitimos a atualização das matrizes *L* e *U* oriundas da fatoração, evitando redecompor a base a cada iteração.
- O emprego de diferentes heurísticas para a seleção da variável a ser ramificada a cada iteração do método *Branch-and-Bound*, bem como para a determinação do nó a ser extraído da lista de nós pendentes.
- O uso de heurísticas para detectar precocemente a infactibilidade de um subproblema, para limitar o número de ramificações para cima e para evitar uma base inicial infactível no método de Lemke.

Nas próximas seções, apresentamos as heurísticas que criamos para acelerar o programa.

4.2 ESCOLHENDO A VARIÁVEL A SER RAMIFICADA

Como visto no algoritmo apresentado na Seção 3.4, é possível que a solução obtida em um nó qualquer não satisfaça a restrição de cardinalidade do problema principal ou algumas variáveis possuam valor não nulo menor que seus limites inferiores. Nestes casos, devemos escolher uma variável para ramificar e incluir dois nós na lista de nós pendentes, um associado à ramificação para cima e outro à ramificação para baixo. Mas como determinar tal variável?

A opção mais simples e intuitiva seria escolher aleatoriamente umas das variáveis não ramificadas. Porém, tal escolha pode não ser eficiente, uma vez que não induz à rápida obtenção de uma solução inteira que sirva de limitante para o algoritmo *Branch-and-Bound*. Desta forma, seria necessário resolver um número muito grande de subproblemas até que obtermos a convergência do algoritmo.

De fato, parece-nos razoável que a escolha da variável a ser ramificada envolva a distância entre o valor atual das variáveis e os seus limites. Imbuídos dessa idéia, elaboramos duas estratégias de ramificação, que apresentamos a seguir.

4.2.1 Ramificando a variável mais próxima de um de seus limites.

A primeira estratégia que nos parece eficiente para a seleção da variável por ramificar consiste em escolher, dentre as variáveis não ramificadas e fora dos seus limites, aquela que está mais próxima do valor zero ou do seu limite inferior.

Em notação matemática, uma vez definido o conjunto $P = \{ j \mid 0 < x_j < l_j \}$, escolhemos a variável x_i tal que

$$i = \arg\min_{j \in P} \{ \min \{ x_j, l_j - x_j \} \}.$$

Essa heurística tem como vantagem provocar a menor alteração possível com relação ao problema do nó pai. Deste modo, a solução ótima obtida para o nó pai será uma boa solução inicial para o novo subproblema, acelerando o método de Lemke.

4.2.2 Ramificando a variável mais distante de um de seus limites.

Uma segunda maneira de determinar a variável a ser ramificada consiste em usar uma estratégia oposta àquela apresentada acima, ou seja, escolher a variável que está mais longe do zero ou do seu limite inferior. Matematicamente, escolhemos a variável x_i tal que

$$i = \arg\min_{j \in P} \left\{ \max \left\{ x_j, l_j - x_j \right\} \right\}.$$

À primeira vista, essa estratégia é análoga à anterior, pois, para cada subproblema, incluímos na lista de nós pendentes tanto a ramificação para cima como a ramificação para baixo. Assim, se em uma dessas ramificações a variável está sendo empurrada para o seu limite mais distante, na outra ela estará se dirigirindo ao seu limite mais próximo. Entretanto, como veremos nos resultados numéricos, quando combinada com a escolha do nó pendente a ser explorado, essa estratégia produz resultados diversos daqueles apresentados para a regra da Seção 4.2.1.

4.2.3 Ramificando a variável mais distante de zero e de seu limite inferior.

Uma outra maneira de determinar a variável a ser ramificada consiste em selecionar, dentre as variáveis não ramificadas e fora dos seus limites, aquela que está mais longe, ao mesmo tempo, do zero e do seu limite inferior. Isso corresponde a escolher a variável x_i tal que

$$i = \arg\max_{j \in P} \left\{ \min \left\{ x_j, l_j - x_j \right\} \right\}.$$

Essa estratégia consiste em atacar diretamente as variáveis mais "problemáticas", ou seja, mais distantes de um valor factível, na esperança de que as demais variáveis sejam empurradas automaticamente pelo algoritmo para o zero ou para seus limites inferiores, fazendo com que caminhemos mais rapidamente para uma solução factível dentro do ramo da árvore que estamos explorando.

Uma alternativa semelhante seria selecionar, dentre as variáveis não ramificadas e fora dos seus limites, aquela que está mais distante do ponto médio do intervalo [0, l_j]. Caso todos

os limitantes l_j sejam idênticos, essa regra é equivalente à escolha da variável mais distante do zero e de seu limite inferior.

4.2.4 Ramificando quando o nó pai tem cardinalidade maior ou igual a *K*.

As estratégias apresentadas acima são eficientes quando a solução do nó pai tem cardinalidade menor que o limite *K*. Caso tenhamos mais componentes positivas que o permitido, parece mais interessante considerar apenas a distância das variáveis ao zero, de modo que possamos eliminar uma delas o mais rápido possivel, caminhando assim em direção à satisfação da restrição $Card(nz(x)) \leq K$.

Tomando como base as estratégias apresentadas nas seções 4.2.1 e 4.2.2, foi possível criar duas estratégias de ramificação diferentes para o caso da solução do nó pai possuir mais que *K* componentes positivas. A primeira, seguindo a idéia da Seção 4.2.1, escolhe ramificar (para baixo) a varável mais perto do zero, ao passo que a segunda, seguindo a idéia da Seção 4.2.2 escolhe ramificar (para baixo) a variável mais distante do zero. Essas duas alternativas serão analisadas no próximo capítulo.

4.3 ESCOLHENDO UM NÓ DA LISTA DE NÓS PENDENTES

Como visto no algoritmo apresentado na Seção 3.4, a cada iteração do algoritmo *Branch-and-Bound*, escolhemos um nó da lista de nós pendentes para analisar.

Sabemos que árvore de soluções pode chegar a ter $2^{N+1} - 1$ nós, de modo que vasculhá-la completamente geraria um gasto computacional enorme. Assim, a estratégia de

seleção do nó pendente é vital para o bom andamento do programa. Apresentamos a seguir as heurísticas que utilizamos em nosso programa.

4.3.1 Escolhendo o nó com melhor ζ.

Em nossa primeira heurística, escolhemos, da lista nós pendentes, aquele nó cujo pai tem função objetivo com o menor valor possível.

Caso ocorra um empate, escolhemos o nó de acordo com o tipo de ramificação da variável. Para exemplificar como isso é feito, suponhamos que dois nós com um pai comum sejam os melhores candidatos da lista de nós pendentes. Suponhamos, também, que tenhamos decidido ramificar a variável em função de sua proximidade com um de seus limites. Neste caso, se a variável estiver próxima do zero, tomamos o nó associado à ramificação para baixo. Por outro lado, escolhemos o nó associado à ramificação para cima caso a variável esteja próxima de seu limite inferior. Uma estratégia análoga é adotada se decidirmos ramificar a variável mais distante de seu limite.

4.3.2 Fazendo uma busca em profundidade.

A idéia da busca em profundidade é achar uma solução inteira factível o mais rápido possível. Assim, podemos iniciar o processo de eliminação dos nós da lista cujo valor da função objetivo é maior que o do nosso limitante.

Nessa heurística, a cada iteração, descemos um nível na árvore do *Branch-and-Bound*, escolhendo sempre, dentre os nós da lista de nós pendentes, aquele com o maior número de variáveis ramificadas (ou seja, com o maior grau).

Mais uma vez, pode ocorrer empate entre dois nós, sendo um referente à ramificação para baixo de uma variável e o outro à ramificação para cima da mesma. Também nesse caso, usamos a estratégia adotada para selecionar a variável a ser ramificada como critério de desempate, escolhendo o nó associado à ramificação mais vantajosa.

4.3.3 Fazendo uma busca em largura.

A idéia da busca em largura é oposta à da busca em profundidade. Nela, a cada iteração, procuramos, na lista de nós pendentes, o nó que tenha o menor grau possível, ou seja, aquele que possua o menor número de ramificações.

Uma das vantagens da busca em largura é que ela pode ser implementada em paralelo. Como os nós de um mesmo nível usam como base inicial a base ótima de seus pais, que pertencem ao nível anterior, podemos dividir a exploração dos nós entre vários processadores ou computadores diferentes, resolvendo, de forma eficiente, vários subproblemas de uma só vez.

Mesmo quando não é possível usar computação paralela, a busca em largura nos permite utilizar a solução de um nó para achar a solução de seu irmão. Em geral, a solução de um irmão é ótima, mas não factível, servindo, em muitos casos, como uma boa solução inicial para o algoritmo de Lemke.

Uma vez que pode ocorrer empate ao escolhermos o nó com o menor grau, utilizamos duas regras distintas de desempate. A primeira consiste em escolher sempre o primeiro (ou o último) nó com o grau mais baixo dentre todos os que fazem parte da lista de nós pendentes. Já a segunda maneira de desempate é similar àquela usada no caso da busca em profundidade, envolvendo a escolha do nó mais vantajoso de acordo com os critérios da Seção 4.2.

4.4 APROVEITANDO A BASE NO MÉTODO DE LEMKE

Uma vez que o subproblema de um nó pai difere do subproblema do nó filho em apenas uma restrição, podemos dizer que a solução do primeiro satisfaz o critério de otimalidade do segundo, não sendo, porém, factível para este. Assim, como o método de Lemke resolve um problema partindo de uma solução complementar, mas não factível, podemos utilizá-lo para resolver o problema do nó filho partindo da base do nó pai, em lugar de adotar a base trivial.

No nosso algoritmo, ao guardar um nó na lista de pendentes, associamos ao mesmo um vetor *vb* que contém os índices das variáveis básicas de seu nó pai. Ao retirarmos um nó da lista de pendentes, criamos a base inicial do nó filho a partir dos índices guardados em *vb*, tornando assim a solução do problema mais rápida e barata.

Entretanto, o reaproveitamento da base traz como conseqüência a necessidade de redefinirmos o vetor h, usado para montar o problema (2.7) e para gerar uma solução factível que inclua a variável z_0.

Quando usamos a base inicial trivial $B = I$, associada apenas às variáveis de folga w, qualquer vetor de componentes estritamente positivas pode ser usado como h. Entretanto, se a base inicial é diferente da matriz identidade, ou seja, se desejamos incluir componentes do vetor z dentre as variáveis básicas da solução inicial, o vetor h fornecido pelo usuário nem sempre é válido. Isto ocorre porque a simples exigência de que $h > 0$ não é suficiente para garantir a não negatividade das variáveis básicas após o primeiro pivoteamento, isto é, após a inclusão da coluna de z_0 na base.

A literatura sugere que recomecemos o método de Lemke a partir de $B = I$ sempre que alguma variável básica da solução inicial for negativa. Entretanto, essa estratégia pode impedir que usemos com freqüência a base ótima do nó pai do subproblema que estamos resolvendo, o que certamente acarretaria um aumento do tempo gasto para a obtenção de sua solução.

Em nosso algoritmo, utilizamos uma heurística que nos permite obter um vetor *h* adequado a partir de um vetor fornecido pelo usuário. Para apresentar essa estratégia, vamos reescrever a equação (2.8) na forma

$$-hz_0 + \tilde{M}\tilde{x} = q\,,$$

em que $\tilde{M} = \begin{bmatrix} -M & I \end{bmatrix}$ e $\tilde{x} = \begin{bmatrix} z^T & w^T \end{bmatrix}^T$. Neste caso, a base inicial fornecida pelo nó pai do problema que temos que resolver é formada por um subconjunto das colunas de $\tilde{M}$, de modo que podemos escrever também

$$-hz_0 + B\tilde{x}_B + N\tilde{x}_N = q\,,$$

onde $\tilde{x}_B$ corresponde ao vetor de variáveis básicas e $\tilde{x}_N$ ao vetor de variáveis não básicas. Como a matriz *B* é não singular e as variáveis não básicas são iguais a zero, devemos ter, simplesmente,

$$\tilde{x}_B = B^{-1}q + B^{-1}hz_0 \geq 0\,.$$

Tomando $z_0 = 1$, podemos forçar $\tilde{x}_B$ a ser igual a um vetor positivo *e* escolhido pelo usuário[3], bastando para isso definir

$$h = Be - q\,.$$

Assim, garantimos que a solução básica formada após o ingresso de z_0 na base seja sempre positiva, característica fundamental para que o método de Lemke funcione. Com isso, evitamos ter que usar a base trivial.

[3] Na prática, usamos o vetor $e = [1, 1, ..., 1]^T$.

Essa estratégia mostrou-se eficiente, reduzindo bastante o tempo consumido pelo nosso método. Entretanto, como a escolha do vetor de componentes positivas é arbitrária, é possível que exista outro vetor capaz de acelerar ainda mais o algoritmo. Infelizmente, não pudemos estudar de forma aprofundada esse tópico, que deixamos para um possível trabalho futuro.

4.5 ATUALIZANDO A FATORAÇÃO LU DA BASE.

Seja $B \in \Re^{mxm}$ a base associada a uma iteração do método de Lemke e sejam $L, U \in \Re^{mxm}$ as matrizes triangulares geradas pela decomposição LU [4] de *B*. Ao final da iteração, uma coluna de *B* precisa ser substituída por outra coluna que, até então, era não básica. Para evitar a refatoração completa da nova base $\overline{B}$, é possível atualizar apenas a matriz *U*, mantendo a matriz *L* triangular inferior. Os principais métodos de atualização da base foram propostos por Bartels [1] e por Forrest e Tomlin [8, 10], no início da década de 70 do século 20. Apresentamos a seguir a atualização de Forrest e Tomlin, que foi incluída em nosso algoritmo.

Suponhamos que a $t^{\text{ésima}}$ coluna de *B* tenha sido alterada. Neste caso, dada a decomposição $B = LU$, podemos escrever a nova base como $\overline{B} = LV$, onde *V* é triangular superior a menos da $t^{\text{ésima}}$ coluna, que é chamada de *espinho*. A atualização da fatoração LU consiste em transformar *V* em uma nova matriz triangular superior através de manipulações algébricas envolvendo permutações de linhas e colunas.

Na atualização proposta por Forrest e Tomlin, primeiramente, permuta-se o espinho com a última coluna da matriz, bem como se permuta a $t^{\text{ésima}}$ linha com a linha final da matriz. Tal processo é realizado através da pré e da pós-multiplicação da matriz *V* por uma matriz de permutação *R*, conforme ilustrado na Figura 4.1.

[4] O processo descrito nesta seção pode ser estendido ao caso em que adotamos a fatoração LU com pivoteamento parcial. Entretanto, preferimos excluir o pivoteamento para simplificar a apresentação.

A nova matriz RVR^T é triangular superior a menos da última linha. Para eliminar os elementos indesejáveis desta linha, aplicamos uma série de pivoteamentos, que são armazenados em uma matriz triangular inferior $\overline{L}$. Ao final do processo, temos a matriz $\overline{U} = \overline{L}RVR^T$, que é triangular superior. Deste modo, podemos escrever $\overline{B} = L\overline{U}$.

Naturalmente, o processo descrito acima pode ser aplicado em iterações sucessivas, desde que guardemos as matrizes envolvidas em cada atualização. Neste caso, a resolução de sistemas que envolvem a matriz $\overline{U}$ torna-se mais cara a cada iteração, de modo que é comum refatorar a base depois de um número fixo de atualizações. Além disso, para evitar instabilidade numérica, B também é refatorada sempre que encontramos um pivô muito pequeno durante a atualização.

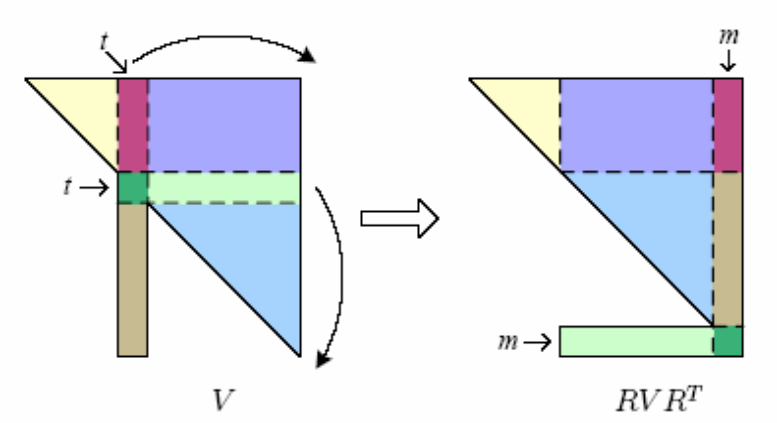


Figura 4.1 – Esquema da atualização de Forrest e Tomlin sobre a matriz *V*, extraído de [10].

4.6 HEURÍSTICAS ADICIONAIS.

Além das heurísticas ligadas aos passos tradicionais do método *Branch-and-Bound*, apresentadas na Seção anterior, procuramos introduzir outras estratégias para aceleração de nosso

algoritmo. Essas heurísticas particulares para o problema com o qual trabalhamos são apresentadas abaixo.

4.6.1 Limitando o número de ramificações para cima.

Uma vez que a restrição de cardinalidade implica na existência de, no máximo, K variáveis não nulas, é absolutamente desnecessário continuar ramificando após executarmos K ramificações para cima, uma vez que, se forçarmos uma nova variável a assumir um valor positivo maior ou igual a seu limite inferior, tornaremos infactível a solução.

Sendo assim, se o nó com o qual trabalhamos já possuir K variáveis ramificadas para cima, todas as outras variáveis devem ser eliminadas (ou seja, devem ser forçadas a valer zero). Neste caso, teremos que resolver um subproblema composto apenas pelas K variáveis selecionadas. Naturalmente, a solução deste subproblema será uma folha da árvore binária, de modo que sua função objetivo servirá como limitante para o *Branch-and-Bound*.

4.6.2 Determinando precocemente a infactibilidade de um subproblema.

Uma vez que, ao ramificarmos uma variável para cima, forçamos esta a assumir ao menos o valor de seu limite inferior, podemos detectar previamente a existência de infactibilidade, caso uma das linhas da matriz A possua apenas componentes positivas.

Sejam, então, C o conjunto de variáveis que já foram ramificadas para cima, l o vetor de limites inferiores das variáveis, A_i uma linha da matriz A que contenha apenas coeficientes positivos e $b_{i,}$ a coordenada do vetor do lado direito, b, correspondente a esta linha. O subproblema que estamos resolvendo certamente será infactível se observarmos que

$$\sum_{j \in C} A_{i,j}\ l_j > b_i\,, \tag{4.1}$$

uma vez que, como sabemos $\sum_{j \in C} A_{i,j}\ l_j < \sum_j A_{i,j} x_j$.

Detectamos esse tipo de infactibilidade no momento em que escolhemos a variável a ser ramificada. Se (4.1) ocorre para alguma restrição, não incluimos a ramificação para cima da variável selecionada na lista de nós pendentes.

CAPÍTULO 5

RESULTADOS COMPUTACIONAIS

5.1 OBJETIVOS

Neste capítulo, descrevemos os experimentos efetuados para investigar a eficiência do método que propusemos para a solução de problemas de otimização na forma (1.6). A análise do desempenho do algoritmo foi feita com base no tempo de execução, no total de iterações do método de Lemke e do número de subproblemas resolvidos no processo do *Branch-and-Bound*. Todos os testes foram feitos utilizando-se o programa Matlab em um computador com processador AMD Sempron 64 bits 1600MHZ e memória principal de 1 GB.

5.2 OS PROBLEMAS DE CARTEIRAS DE INVESTIMENTO

Para testar a eficiência de nossa programa, utilizamos o conjunto de problemas-teste[5] selecionados por Chang *et al.* [5]. Os cinco problemas selecionados correspondem a indicadores econômicos baseados em grandes bolsas de valores do mundo. A primeira instância, denominada Hang Seng, tem 31 ativos e está relacionada à bolsa de Hong Kong. A segunda, DAX, com 85 ativos, está associada à bolsa alemã. O terceiro grupo de problemas, relativo ao mercado financeiro do Reino Unido, é denominado FTSE e tem com 89 ativos. A quarta instância é a americana S&P, com 98 ativos. A última, denominada Nikkei, contém 225 ativos da bolsa japonesa.

Para cada instância, geramos problemas diferentes variando o termo de penalização λ de (1.5). Vale lembrar ainda que as matrizes de covariância relcionadas as cinco instâncias são definidas positivas, logo a solucão de cada subproblema será única. Originalmente, restringimos a 10 o número de ativos que compõem a carteira (ou seja, tomamos $K = 10$) e utilizamos o limite mínimo de investimento de 1 % para cada ativo (o que significa que $l_i = 0{,}01$, $i = 1, ..., N$). Entretanto, como veremos abaixo, para tornar alguns testes mais desafiadores, foi preciso usar valores mais altos para K e l_i.

5.3 TESTANDO AS VARIANTES DO MÉTODO *BRANCH-AND-BOUND*

No Capítulo 4, definimos diferentes maneiras de percorrer a árvore gerada pelo algoritmo *Branch-and-Bound* implícito. Nesta seção, compararamos as alternativas apresentadas, de modo a determinar a combinação mais promissora para o nosso problema específico.

[5] Os dados dos problemas estão disponíveis em http://people.brunel.ac.uk/~mastjjb/jeb/orlib/portinfo.html.

Como apresentado na Seção 4.2, existem diferentes maneiras de se escolher a variável a ser ramificada. Em nossos testes, analisamos quatro possibilidades:

1. ramificar a variável mais perto de um de seus limites (conforme a Seção 4.2.1);
2. ramificar a variável mais distante de um de seus limites (vide Seção 4.2.2);
3. ramificar a variável mais distante de seus dois limites (apresentada na Seção 4.2.3);
4. escolher aleatoriamente a variável a ser ramificada (estratégia usada como referência na avaliação das demais).

Outra escolha intrínseca ao processo do *Branch-and-Bound* é a do nó a ser selecionado da lista de nós pendentes a cada iteração. As estratégias que analisamos neste caso envolvem[6]:

1. escolher sempre o nó com melhor ζ, como apresentado na Seção 4.3.1;
2. realizar uma busca em profundidade, conforme a Seção 4.3.2;
3. selecionar um nó de forma aleatória (alternativa usada como referência para a análise das demais estratégias).

Para comparar as diversas estratégias de ramificação e de seleção de nó, é necessário escolher problemas que exijam um esforço computacional entre médio e alto. Como sabemos que o número de iterações do método *Branch-and-Bound* é maior para valores de λ perto de 1, ou seja, quando temos o risco como prioridade e há necessidade de diversificar o *portfolio*, decidimos trabalhar apenas com $\lambda \geq 0{,}88$. Além disso, decidimos trabalhar, nessa primeira etapa de análise dos resultados, apenas com os problemas FTSE e S&P, pois estas foram as instâncias que, nos testes, exigiram mais iterações do algoritmo. Para cada instância FTSE, definimos onze problemas, tomando $\lambda \in \{0{,}88; 0{,}89; 0{,}90; ...; 0{,}98\}$. Não usamos valores mais altos de λ para evitar gerar carteiras nas quais o montante aplicado é muito baixo.

[6] Testes preliminares indicaram que a busca em largura não é apropriada para a solução de nosso problema, de modo que não incluímos esta alternativa em nossos resultados.

5.3.1 Determinando a melhor estratégia de ramificação

Em nosso primeiro teste, comparamos os critérios de ramificação das variáveis, usando $K = 10$ e l_i=0,01 e fixando a escolha do nó com menor ζ como estratégia de seleção do nó pendente. Os gráficos das Figura 5.1 e 5.2 mostram os resultados obtidos para as instâncias FTSE e S&P, respectivamente. Nestas figuras, o termo Crit 1 se refere a todas as três estratégias de ramificação baseadas na distância das variáveis ao zero ou aos seus limites inferiores, enquanto Crit 2 representa a escolha aleatória da variável a ser ramificada. As três estratégias foram agrupadas em uma única curva porque não houve diferença entre os resultados por elas alcançados.

Em todos os gráficos desse capítulo, usamos o tempo de execução do nosso algoritmo como parâmetro de comparação. Além disso, para tornar mais clara a vantagem de um método sobre os demais, o eixo vertical dos gráficos indica a razão entre o tempo gasto pelo algoritmo associado a uma curva e o menor tempo obtido dentre todas as estratégias testadas. Assim, observamos, na Figura 5.1, que a função associada a Crit 1 vale sempre 1, indicando que as estratégias baseadas na distância das variáveis a zero e a seus limites inferiores obtiveram os melhores resultados para todos os valores de λ. Por outro lado, a escolha aleatória da variável a ser ramificada consumiu de 1,3 a 3 vezes o tempo da melhor estratégia, como mostra a curva de Crit 2.

A Figura 5.2 mostra que, também para a instância S&P, o desempenho das estratégias baseadas na distância das variáveis a 0 e a l_i é indiscutivelmente superior ao da escolha aleatória, como previámos.

O motivo de as três estratégias propostas na Seção 4.2 terem apresentado rigorosamente o mesmo comportamento quando usamos $K = 10$ e $l_i = 0{,}01$ está associado ao fato de haver uma tendência à diversificação quando λ é alto. Como nosso algoritmo trata a restrição de cardinalidade de forma implícita, os nós dos primeiros níveis da árvore do *Branch-and-Bound* (ou seja, aqueles níveis nos quais o número de variáveis fixadas é pequeno) costumam conter um número de ativos não nulos superior a K. Quando isso acontece, é preciso simplesmente cortar

ativos da carteira, como comentado na Seção 4.2.4, de modo que a estratégia de ramificação usual tem pouca importância.

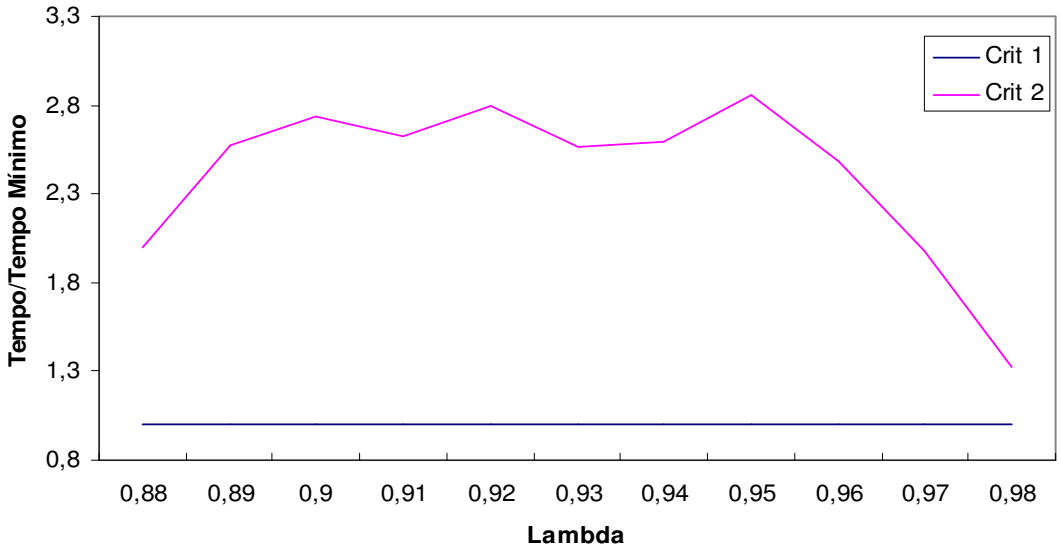


Figura 5.1 – Comparação das estratégias de ramificação para a instância FTSE, fixando a seleção do nó com menor ζ como estratégia de escolha do nó pendente.

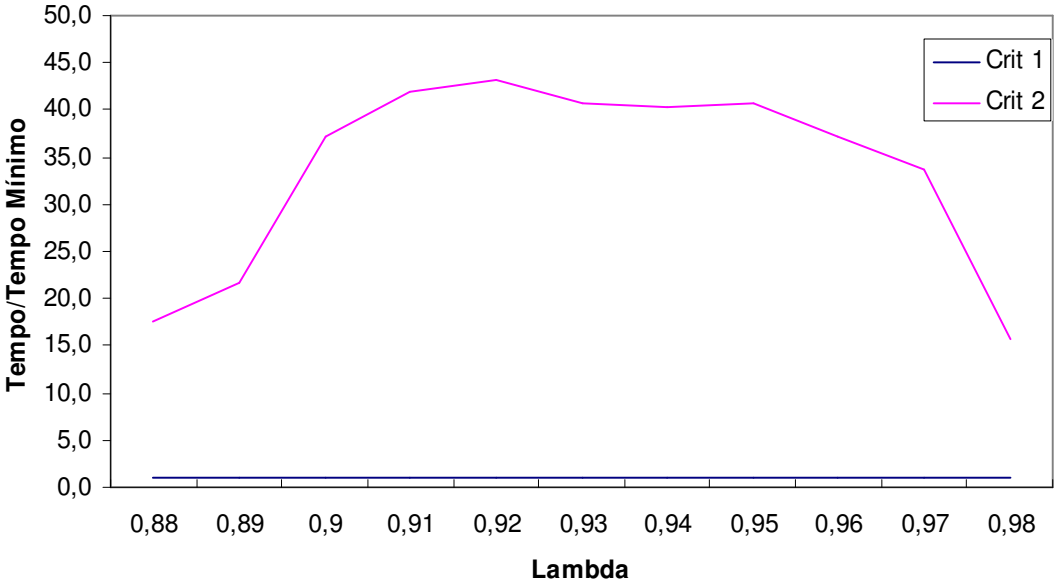


Figura 5.2 – Comparação das estratégias de ramificação para a instância S&P, fixando a seleção do nó com menor ζ como estratégia de escolha do nó pendente.

Em virtude dos resultados alcançados, para que a análise das estratégias de ramificação pudesse ser feita, foi necessário refazer os testes com outros valores para K e l_i. Os valores escolhidos foram $K = 15$ e $l_i = 0{,}12$; $i = 1, ..., N$.

Nas Tabelas 5.1 e 5.2, apresentamos os resultados obtidos quando a seleção do nó pendente é baseada no valor de ζ. Nessas tabelas, a coluna **Iter** indica o número de nós percorridos pelo algoritmo *Branch-and Bound*, a coluna **ItLemke** fornece o número total de iterações executadas pelo algoritmo de Lemke e a coluna **Tempo** indica o tempo de execução (em segundos) comsumido por cada problema. Para os demais testes desta subseção, o critério 1 refere-se à ramificação da variável mais distante de um de seus limites, o critério 2 à ramificação da variável mais próxima de um de seus limites, e o critério 3 refere-se à ramificação da variável mais distante de seus dois limites. Como se observa, em nossos novos experimentos, desprezamos a ramificação aleatória, pois ele apresentou resultados muito inferiores às demais.

Tabela 5.1 – Desempenho do algoritmo para a instância FTSE, fixando a seleção do nó com menor ζ como estratégia de escolha do nó pendente e variando a estratégia de ramificação.

λ	Critério 1			Critério 2			Critério 3		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	276	585	2,15	276	585	2,26	46	110	0,42
0,89	364	710	2,75	364	710	2,91	42	116	0,40
0,90	366	601	3,03	366	601	2,76	62	142	0,54
0,91	284	448	2,14	284	448	2,23	106	264	0,89
0,92	220	395	1,77	220	395	1,60	168	422	1,42
0,93	244	496	1,91	244	496	1,79	220	626	1,92
0,94	116	250	0,98	116	250	0,87	152	467	1,33
0,95	98	205	0,74	98	205	0,70	126	448	1,12
0,96	110	246	0,84	110	246	0,81	156	604	1,38
0,97	224	547	1,33	224	547	1,35	184	789	1,58
0,98	178	510	0,95	178	510	0,89	212	1373	1,56

Tabela 5.2 – Desempenho do algoritmo para a instância S&P, fixando a seleção do nó com menor ζ como estratégia de escolha do nó pendente e variando a estratégia de ramificação.

λ	Critério 1			Critério 2			Critério 3		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	602	1847	5,54	602	1847	5,57	387	1352	4,12
0,89	796	2534	7,51	796	2534	7,63	560	2088	6,34
0,90	816	2865	8,42	816	2865	8,33	670	2574	7,47
0,91	530	1933	6,25	530	1933	6,14	515	1918	6,06
0,92	728	2381	8,14	728	2381	8,31	594	2136	7,00
0,93	1458	4338	15,06	1458	4338	15,07	978	3640	11,61
0,94	1048	3365	11,32	1048	3365	11,13	742	2967	8,77
0,95	1174	4103	11,85	1174	4103	11,90	748	3465	9,04
0,96	1012	4148	10,40	1012	4148	10,39	588	3384	7,45
0,97	1228	5754	11,56	1228	5754	11,15	772	5392	8,42
0,98	450	2506	3,87	450	2506	3,91	338	2660	3,77

As Figuras 5.3 e 5.4 apresentam os resultados obtidos para as instâncias FTSE e S&P.

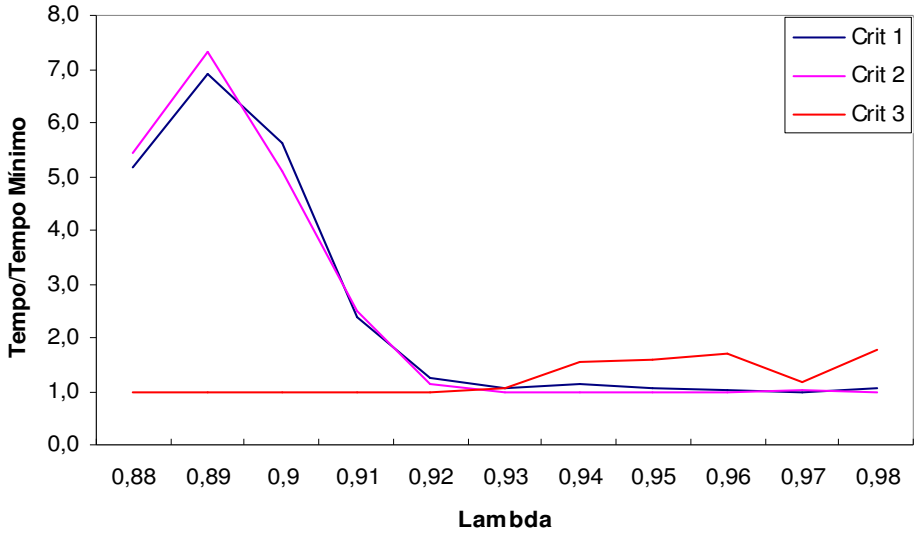


Figura 5.3 – Comparação das estratégias de ramificação para a instância FTSE, fixando a seleção do nó com menor ζ como estratégia de escolha do nó pendente.

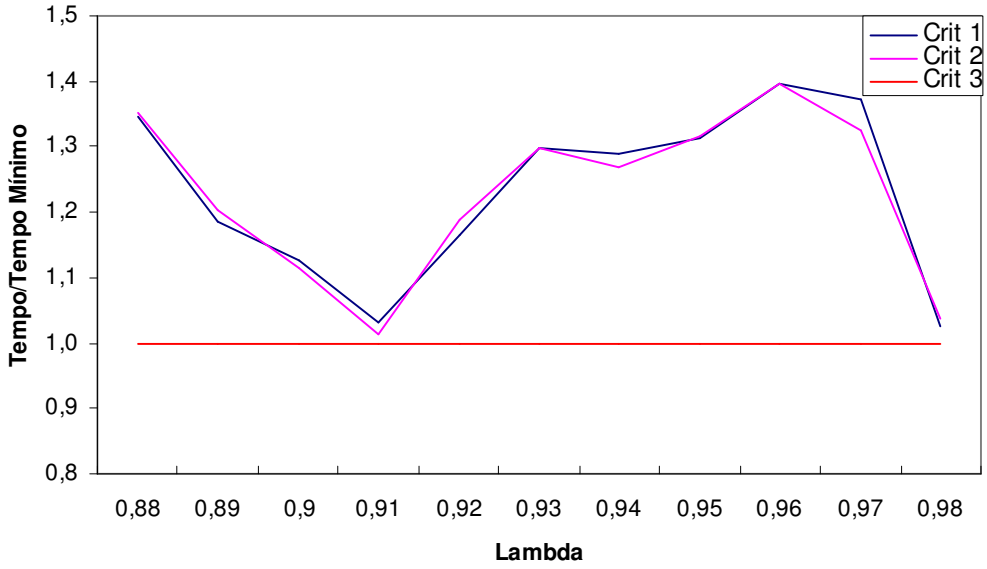


Figura 5.4 – Comparação das estratégias de ramificação para a instância S&P, fixando a seleção do nó com menor ζ como estratégia de escolha do nó pendente.

As Tabelas 5.3 e 5.4, e as Figuras 5.5 e 5.6, apresentam os resultados para o caso em que a escolha do nó pendente é feita através de uma busca em profundidade.

Tabela 5.3 – Desempenho do algoritmo para a instância FTSE, usando a busca em profundidade como estratégia de escolha do nó pendente e variando a estratégia de ramificação.

λ	Critério 1			Critério 2			Critério 3		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	2194	4403	17,39	278	589	2,31	76	172	0,72
0,89	1726	3180	14,05	364	710	2,77	86	205	0,90
0,90	1428	2285	11,53	368	606	2,82	80	169	0,80
0,91	2570	4218	20,56	286	450	2,19	126	303	1,30
0,92	1318	2255	10,93	220	395	1,61	164	407	1,71
0,93	1601	2902	12,80	244	496	1,77	248	621	2,46
0,94	1092	2034	8,60	116	250	0,88	194	605	1,95
0,95	1018	2011	7,75	98	205	0,70	280	979	2,66
0,96	848	1772	6,38	110	246	0,81	186	677	1,90
0,97	802	1869	5,38	224	547	1,34	1252	5989	7,84
0,98	570	1449	3,26	178	510	0,85	334	1788	2,25

Tabela 5.4 – Desempenho do algoritmo para a instância S&P, usando a busca em profundidade como estratégia de escolha do nó pendente e variando a estratégia de ramificação.

λ	Critério 1			Critério 2			Critério 3		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	14821	31354	108,90	1867	3939	14,97	634	1323	5,90
0,89	19341	43597	132,59	2670	5954	21,12	938	1950	8,12
0,90	37072	88052	233,80	3149	7267	24,42	1098	2397	9,38
0,91	34398	74272	222,96	2796	5989	22,85	878	1876	8,18
0,92	41776	82902	250,21	3850	7375	29,76	1088	2231	9,50
0,93	17273	33267	115,29	7238	13742	52,64	2282	5649	20,38
0,94	16573	33375	105,33	6064	12000	41,91	2588	7497	22,38
0,95	12170	25507	75,84	5042	10700	34,06	4582	16221	35,67
0,96	11360	25508	64,82	3038	7128	20,17	4398	18638	26,93
0,97	4920	12177	26,85	2344	6118	13,28	1934	9522	12,55
0,98	1460	4247	7,73	568	1732	3,11	516	2961	3,91

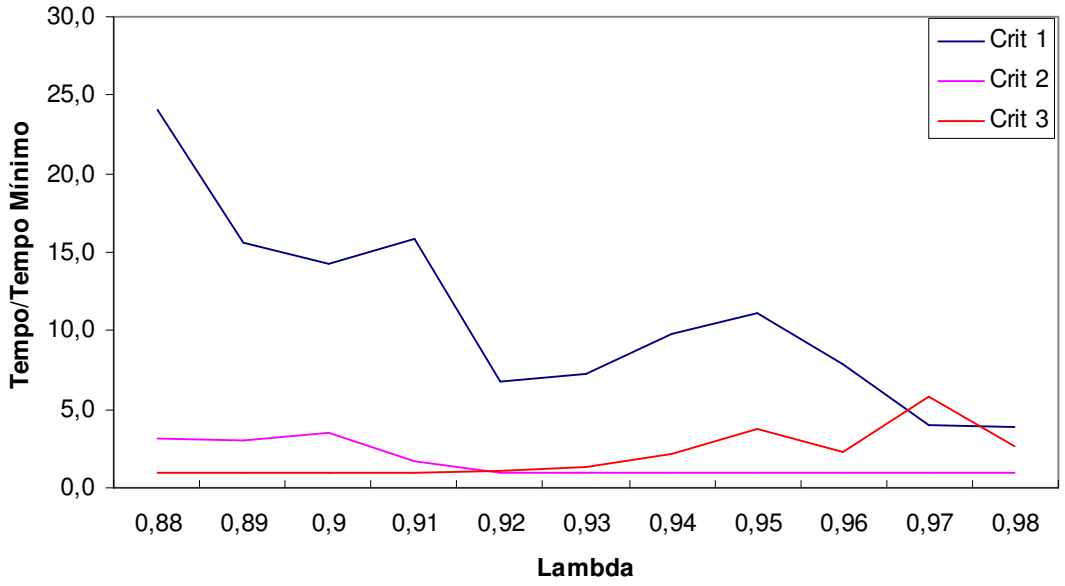


Figura 5.5 – Comparação das estratégias de ramificação para a instância FTSE, usando a busca em profundidade como estratégia de escolha do nó pendente.

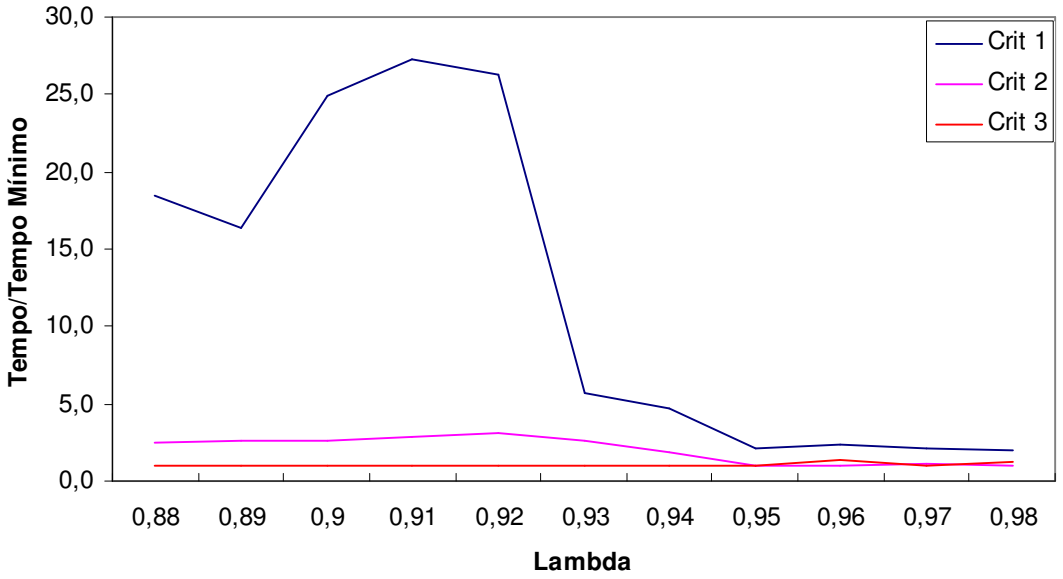


Figura 5.6 – Comparação das estratégias de ramificação para a instância S&P, usando a busca em profundidade como estratégia de escolha do nó pendente.

Os resultados obtidos quando o nó pendente selecionado é aquele que apresenta o menor valor de ζ indicam uma certa superioridade da estratégia de ramificação da variável mais distante de seus dois limites, apesar das duas outras estratégias terem apresentado um desempenho melhor para os dois últimos valores de λ testados para a instância FTSE.

Quando se faz uma busca em profundidade na árvore do *Branch-and-Bound*, percebemos uma nítida vantagem da estratégia de ramificação da variável mais próxima de um de seus limites (Crit 2) sobre a ramificação da variável mais distante de um de seus limites (Crit 1). Além disso, há uma pequena superioridade da seleção da variável mais distante de seus dois limites sobre a ramificação da variável mais próxima de um de seus limites, embora esta última tenha apresentado um desempenho superior quando resolvemos a instância FTSE com $\lambda \geq 0{,}92$.

De uma forma geral, a combinação entre ramificar a variável mais longe de seus dois limites e selecionar o nó com o menor valor de ζ forneceu os melhores resultados.

5.3.2 Determinando a estratégia de ramificação quando temos mais que *K* ativos

Como foi dito na Seção 4.2.4, quando o nó pai tem mais do que K variáveis positivas, o único objetivo do método deve ser a redução do número de ativos da carteira. Assim, a estratégia de ramificação deve levar em conta apenas a distância entre o valor atual da variável e o zero.

Analisamos duas estratégias de seleção da variável a ramificar neste caso. A primeira, que chamamos de Critério 1, consiste em escolher a variável mais distante do zero. A segunda, denominada Critério 2, consiste em fazer o oposto, ou seja, selecionar a variável mais próxima do zero. Uma vez que testes preliminares mostraram que a eficiência de tal escolha está diretamente ligada à estratégia de seleção do nó pendente, decidimos analisar separadamente a combinação desses critérios com a seleção do nó com menor ζ e com a busca em profundidade. No caso em que temos menos que K ativos na solução, adotamos sempre a estratégia de ramificação da variável mais distante de seus dois limites. Restringimos nossos testes à instância S&P, por ser aquela na qual visitamos mais nós da árvore do *Branch-and-Bound*, e usamos $K = 15$ e $l_i = 0{,}12$, $i = 1, ..., N$. As Tabelas 5.5 e 5.6 e as Figuras 5.7 e 5.8 mostram os resultados obtidos.

Tabela 5.5 – Desempenho do algoritmo para a instância S&P, fixando a seleção do nó com menor ζ como estratégia de escolha do nó pendente e variando a estratégia de ramificação quando há mais que K ativos.

λ	Critério 1			Critério 2		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	387	1352	4,12	598	1223	5,49
0,89	560	2088	6,34	888	1832	8,32
0,90	670	2574	7,47	1094	2389	10,11
0,91	515	1918	6,06	878	1876	8,39
0,92	594	2136	7,00	902	1781	8,70
0,93	978	3640	11,61	1798	4193	21,00
0,94	742	2967	8,77	1750	4713	19,22
0,95	748	3465	9,04	1818	5801	19,85
0,96	588	3384	7,45	1272	4918	13,41
0,97	772	5392	8,42	1092	5472	11,02
0,98	338	2660	3,77	354	2072	3,04

Tabela 5.6 – Desempenho do algoritmo para a instância S&P, fixando a busca em profundidade como estratégia de escolha do nó pendente e variando a estratégia de ramificação quando há mais que *K* ativos. O simbolo * indica que foi atingido o limite de 600 segundos sem que o algoritmo chegasse à solução ótima.

λ	Critério 1			Critério 2		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	*	*	*	634	1323	5,90
0,89	*	*	*	938	1950	8,12
0,90	*	*	*	1098	2397	9,38
0,91	*	*	*	878	1876	8,18
0,92	*	*	*	1088	2231	9,50
0,93	*	*	*	2282	5649	20,38
0,94	52362	241495	406,36	2588	7497	22,38
0,95	20572	106250	159,04	4582	16221	35,67
0,96	7128	40844	53,84	4398	18638	26,93
0,97	2136	14499	17,61	1934	9522	12,55
0,98	490	3564	5,05	516	2961	3,91

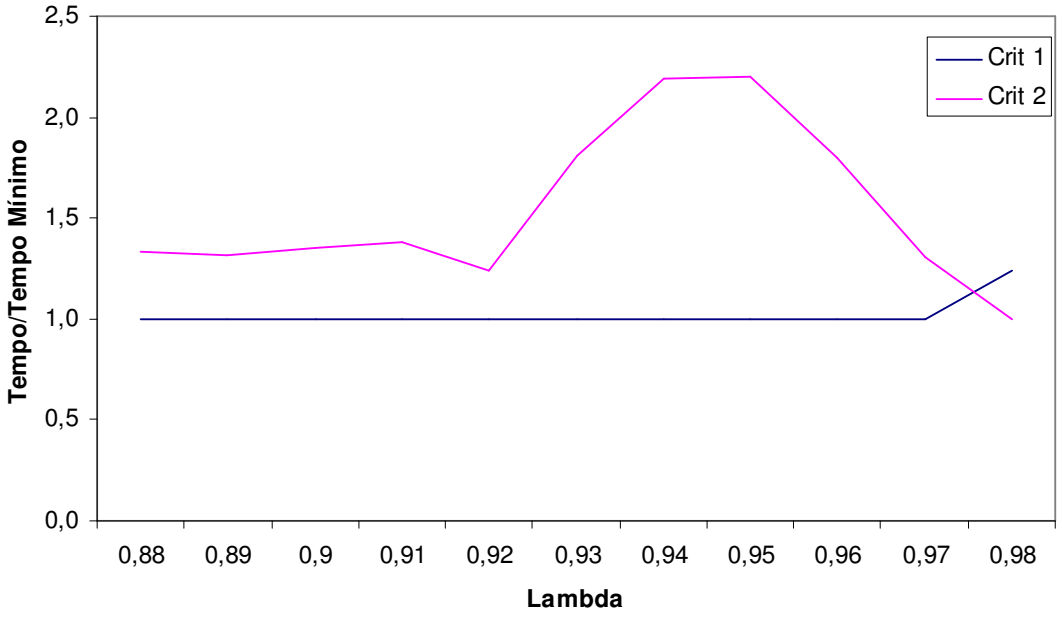


Figura 5.7 – Comparação das estratégias de ramificação quando há mais de *K* ativos na carteira, para a instância S&P, selecionando o nó pendente com menor ζ.

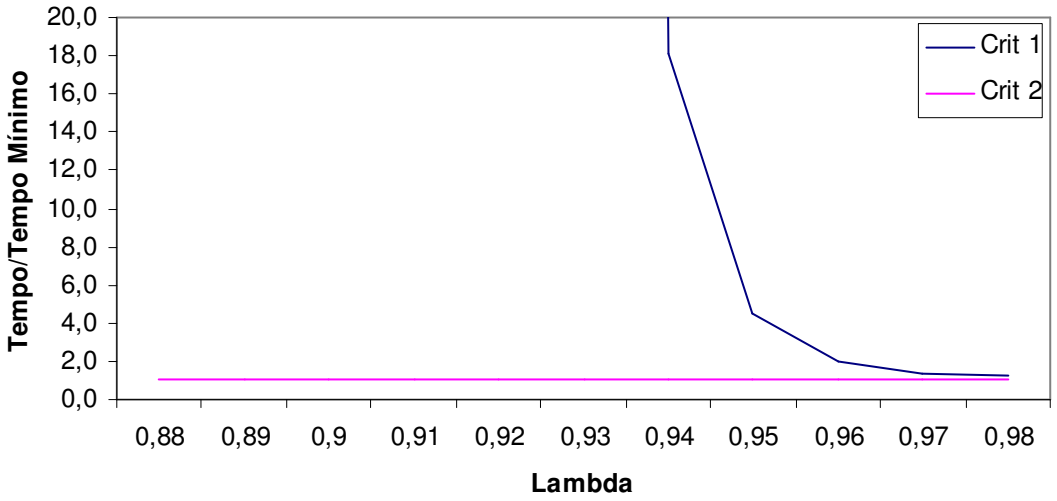


Figura 5.8 – Comparação das estratégias de ramificação quando há mais de K ativos na carteira, para a instância S&P, usando a busca em profundidade.

As Figuras 5.7 e 5.8 mostram as curvas obtidas para cada uma das duas estratégias de seleção do nó. Como se pode notar na Figura 5.7, é relativemente grande a vantagem da estratégia de seleção da variável mais distante do zero quando fixamos a escolha do nó pendente com o menor ζ. Por outro lado, observamos o oposto quando usamos a busca em profundidade, como mostra a Figura 5.8. Neste caso, o tempo gasto com a estratégia de seleção da variável mais distante do zero foi muito alto, passando de 10 minutos para $0{,}88 \leq \lambda \leq 0{,}93$.

Sendo assim, em todos os demais testes deste capítulo, sempre que a escolha do nó pendente foi feita com base no valor de ζ, ramificamos a variável mais distante do zero quando a solução do nó pai possuia mais que K ativos. Já para a busca em profundidade, a variável ramificada se nó pai tinha mais que K ativos foi sempre aquela mais próxima do zero.

5.3.3 Determinando a melhor estratégia de seleção do nó pendente

Concluindo a seção sobre as variantes do método *Branch-and-Bound*, investigamos agora as estratégias de escolha do nó pendente. Mais uma vez, adotamos $K = 15$ e $l_i = 0{,}12$, $i = 1, ..., N$, para destacar o papel dessas estratégias, bem como para minimizar a influência do critério de ramificação das variáveis quando temos mais que K ativos na carteira.

Nas tabelas e gráficos abaixo, usamos o termo Estratégia 1 para representar a seleção do nó pendente com menor ζ e o termo Estratégia 2 para representar a busca em profundidade. A Estratégia 3 corresponde à escolha aleatória de um nó da lista de nós pendentes. Essa última estratégia foi incluída nos testes apenas para indicar quão efetivas são as estratégias de seleção do nó.

As Tabelas 5.7 e 5.8 e as Figuras 5.9 e 5.10 mostram os resultados obtidos quando a ramificação de variáveis é feita tomando a variável mais próxima de um de seus dois limites.

Tabela 5.7 – Desempenho do algoritmo para a instância FTSE, ramificando a variável mais próxima de um de seus limites e variando a estratégia de seleção do nó pendente.

λ	Estratégia 1			Estratégia 2			Estratégia 3		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	276	585	2,26	278	589	2,31	362	799	2,90
0,89	364	710	2,91	364	710	2,77	461	971	3,81
0,90	366	601	2,76	368	606	2,82	500	911	4,02
0,91	284	448	2,23	286	450	2,19	370	651	2,97
0,92	220	395	1,60	220	395	1,61	315	657	2,51
0,93	244	496	1,79	244	496	1,77	313	672	2,42
0,94	116	250	0,87	116	250	0,88	187	435	1,48
0,95	98	205	0,70	98	205	0,70	161	378	1,26
0,96	110	246	0,81	110	246	0,81	176	443	1,35
0,97	224	547	1,35	224	547	1,34	289	744	1,84
0,98	178	510	0,89	178	510	0,85	251	731	1,28

Tabela 5.8 – Desempenho do algoritmo para a instância S&P, ramificando a variável mais próxima de um de seus limites e variando a estratégia de seleção do nó pendente.

λ	Estratégia 1			Estratégia 2			Estratégia 3		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	602	1847	5,57	1867	3939	14,97	3087	10321	29,87
0,89	796	2534	7,63	2670	5954	21,12	3792	13394	37,45
0,90	816	2865	8,33	3149	7267	24,42	4235	16578	42,23
0,91	530	1933	6,14	2796	5989	22,85	4097	16135	41,12
0,92	728	2381	8,31	3850	7375	29,76	3531	13576	35,27
0,93	1458	4338	15,07	7238	13742	52,64	2118	6656	23,52
0,94	1048	3365	11,13	6064	12000	41,91	1688	5854	18,97
0,95	1174	4103	11,90	5042	10700	34,06	1954	8226	20,36
0,96	1012	4148	10,39	3038	7128	20,17	1735	8125	17,32
0,97	1228	5754	11,15	2344	6118	13,28	1888	8904	16,14
0,98	450	2506	3,91	568	1732	3,11	693	3781	5,71

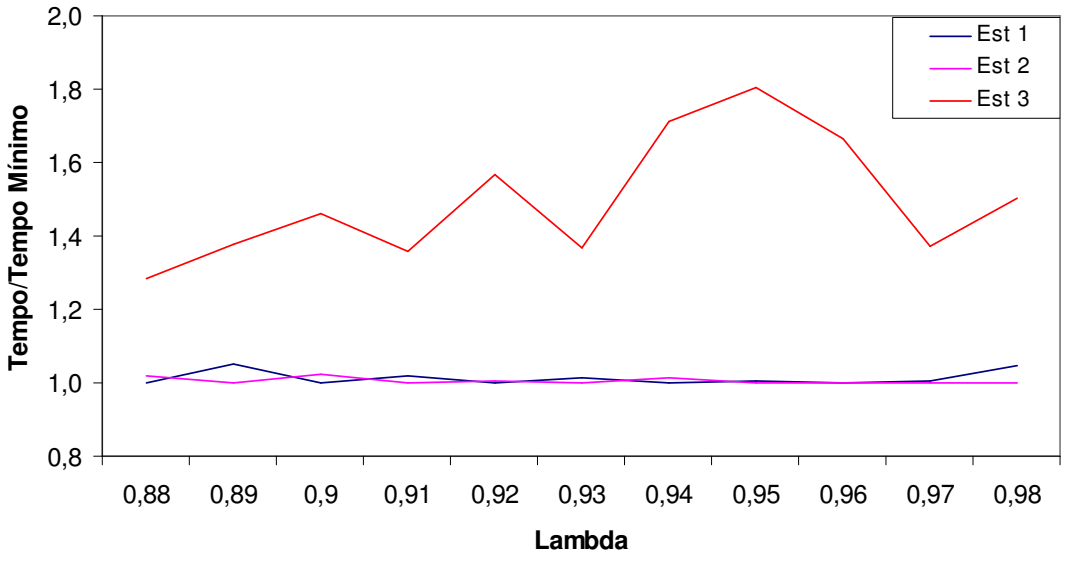


Figura 5.9 – Comparação das estratégias de seleção do nó pendente para a instância FTSE, ramificando a variável mais próxima de um de seus dois limites.

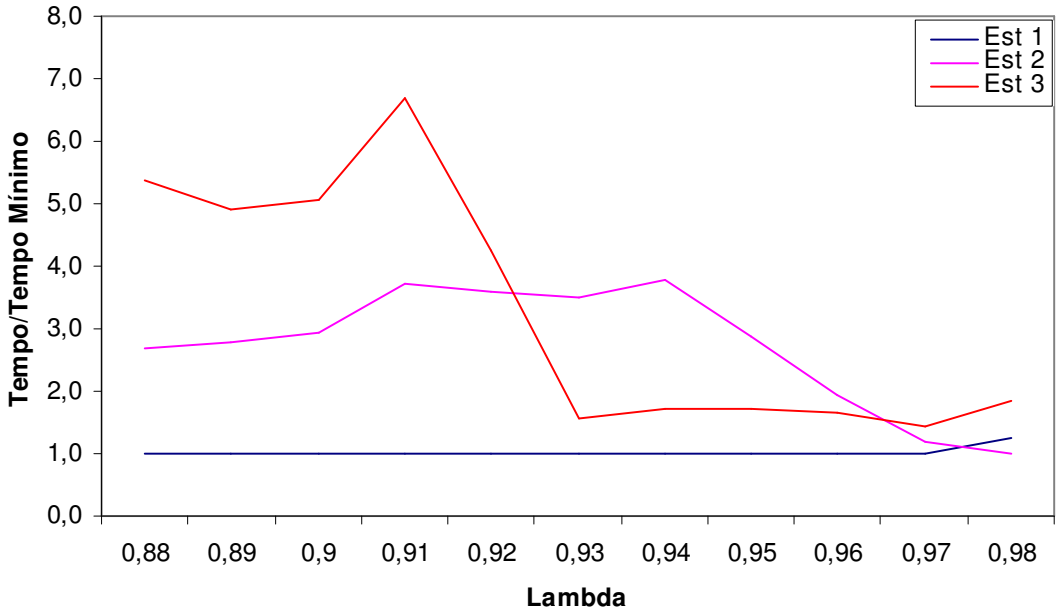


Figura 5.10 – Comparação das estratégias de seleção do nó pendente para a instância S&P, ramificando a variável mais próxima de um de seus dois limites.

Como vemos na Figura 5.9, para a instância FTSE, as duas estratégias de seleção do nó estudadas são equivalentes e superam, em muito, a seleção aleatória. Entretanto, para a instância S&P, a seleção do nó com menor ζ apresenta uma clara vantagem sobre a busca em profundidade, que perde para a estratégia aleatória quando $0{,}93 \leq \lambda \leq 0{,}96$.

Vejamos, agora, o que acontece quando a estratégia de ramificação das variáveis é aquela na qual se escolhe a variável mais distante de seus dois limites. Os resultados, neste caso, são apresentados nas Tabelas 5.9 e 5.10, e nas Figuras 5.11 e 5.12.

Tabela 5.9 – Desempenho do algoritmo para a instância FTSE, ramificando a variável mais distante de seus dois limites e variando a estratégia de seleção do nó pendente.

λ	Estratégia 1			Estratégia 2			Estratégia 3		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	46	110	0,42	76	172	0,72	133	338	1,04
0,89	42	116	0,40	86	205	0,90	161	446	1,38
0,90	62	142	0,54	80	169	0,80	171	402	1,46
0,91	106	264	0,89	126	303	1,30	198	498	1,70
0,92	168	422	1,42	164	407	1,71	212	551	1,80
0,93	220	626	1,92	248	621	2,46	337	943	2,88
0,94	152	467	1,33	194	605	1,95	370	1288	3,12
0,95	126	448	1,12	280	979	2,66	353	1355	2,83
0,96	156	604	1,38	186	677	1,90	219	791	1,84
0,97	184	789	1,58	1252	5989	7,84	486	2443	3,42
0,98	212	1373	1,56	334	1788	2,25	426	2450	2,78

Tabela 5.10 – Desempenho do algoritmo para a instância S&P, ramificando a variável mais distante de seus dois limites e variando a estratégia de seleção do nó pendente.

λ	Estratégia 1			Estratégia 2			Estratégia 3		
	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo	Iter	ItLemke	Tempo
0,88	387	1352	4,12	634	1323	5,90	2940	10843	27,41
0,89	560	2088	6,34	938	1950	8,12	3523	13785	33,11
0,90	670	2574	7,47	1098	2397	9,38	3890	16769	37,52
0,91	515	1918	6,06	878	1876	8,18	3616	15482	35,94
0,92	594	2136	7,00	1088	2231	9,50	2968	12467	30,36
0,93	978	3640	11,61	2282	5649	20,38	1868	7600	20,63
0,94	742	2967	8,77	2588	7497	22,38	1622	7074	18,10
0,95	748	3465	9,04	4582	16221	35,67	1721	8818	17,39
0,96	588	3384	7,45	4398	18638	26,93	1356	8043	13,78
0,97	772	5392	8,42	1934	9522	12,55	1304	8682	12,88
0,98	338	2660	3,77	516	2961	3,91	600	4397	5,67

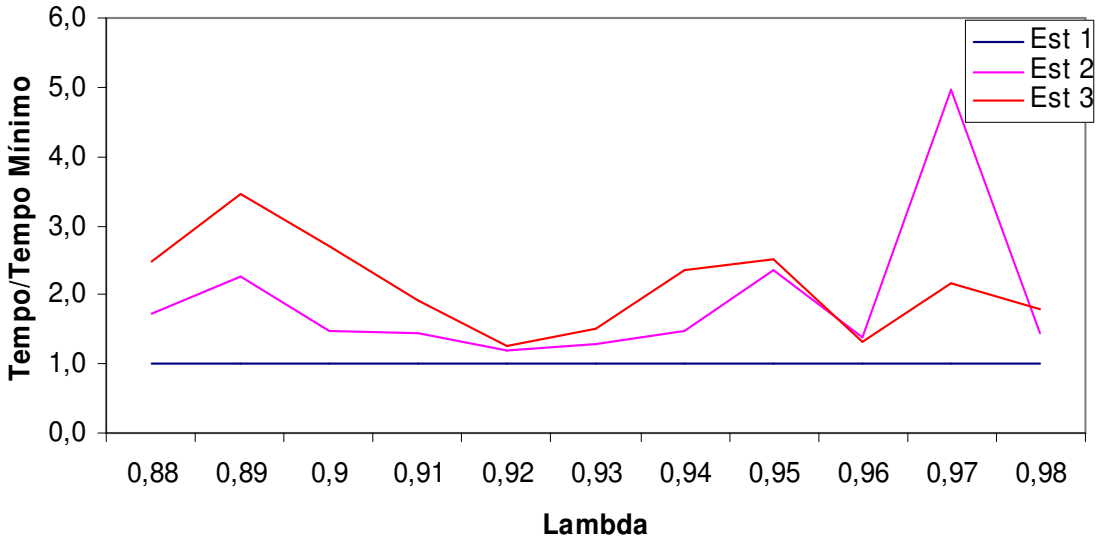


Figura 5.11 – Comparação das estratégias de seleção do nó pendente para a instância FTSE, ramificando a variável mais distante de seus dois limites.

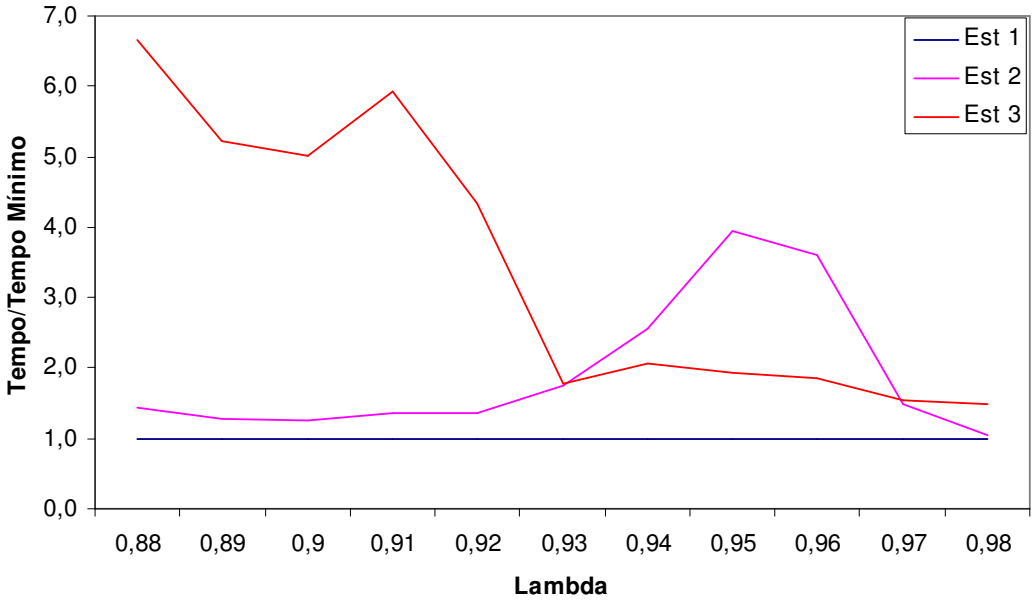


Figura 5.12 – Comparação das estratégias de seleção do nó pendente para a instância S&P, ramificando a variável mais distante de seus dois limites.

Analisando os gráficos acima, observamos que, mais uma vez, a seleção do nó com o menor valor de ζ forneceu os melhores resultados. As Tabelas 5.9 e 5.10 também revelam que, apesar da busca em profundidade ter consumido menos iterações em 6 dos 22 problemas, isso não foi suficiente para que ela obtivesse o menor tempo nesses casos. De fato, a busca em profundidade chegou a perder para a seleção aleatória do nó pendente em cinco dos 22 testes. Em um desses problemas, o *Branch-and-Bound* associado à busca em profundidade chegou a visitar 7,5 vezes mais nós que o algoritmo com a estratégia de seleção do nó baseada no valor de ζ.

5.4 ATUALIZANDO A FATORAÇÃO LU DA BASE

Uma das maneiras usuais de acelerar o método de Lemke consiste em atualizar a fatoração LU da base, em lugar de redecompô-la a cada mudança de base como descrito na Seção 4.5. Como a fatoração domina o custo por iteração, a atualização costuma reduzir acentuadamente o tempo gasto pelo algoritmo. Neste nosso trabalho, adotamos a atualização de Forrest e Tomlin [8, 10], apresentada na Seção 4.5.

Para avaliar a eficiência da atualização, empregamos dois tipos de testes. O primeiro consiste em resolver diretamente problemas de complementariedade linear através do método de Lemke. No segundo teste, resolvemos problemas de programação quadrática inteira mista na forma (1.5), como aqueles empregados na seção anterior.

5.4.1 Aplicando a atualização da base à resolução de problemas de complementaridade

Para avaliar a influência direta da atualização da base sobre o desempenho do método de Lemke, selecionamos dois problemas de programação quadrática (no formato (2.1))

relacionados ao nó raiz da árvore binária gerada ao aplicarmos o método *Branch-and-Bound* à otimização de carteiras de ações. O primeiro teste foi criado tomando o problema com $\lambda = 0{,}9$ da instância FTSE. O segundo teste foi obtido usando o problema com $\lambda = 0{,}9$ da instância S&P.

Os resultados obtidos estão resumidos na Tabela 5.11, que apresenta o tempo (em segundos) consumido tanto pela versão do método de Lemke com uma decomposição LU por iteração como para a versão com a atualização da base.

Tabela 5.11 – Desempenho do método de Lemke com e sem a atualização da decomposição LU.

Decomposição LU	Tempo (s)	
	Problema 1	Problema 2
Sem atualização	0,04	0,03
Com atualização	0,17	0,05

Analisando a tabela, reparamos que, em ambos os testes, o cálculo da decomposição LU da base a cada iteração foi mais eficiente que sua atualização. Para o primeiro problema, o uso da atualização tornou o método de Lemke cerca de quatro vezes mais caro. Já para o segundo problema, o tempo consumido pela versão sem a atualização é 40% menor.

Esse resultado é surpreendente, uma vez que, em teoria, a atualização da fatoração LU consome menos operações aritméticas que a redecomposição da base. A explicação do fenômeno está relacionada ao fato de termos desenvolvido nosso algoritmo de atualização usando o Matlab. De fato, em seu estudo sobre a implementação do método Simplex, Morgan [15] já havia demonstrado que, como o Matlab possui uma série de rotinas especiais para a decomposição da base, é praticamente impossível criar uma rotina de atualização que seja competitiva. Assim, nosso primeiro teste acabou servindo apenas para comprovar que aquilo que valia para o método Simplex também se aplica ao método de Lemke.

Naturalmente, essa desvantagem da atualização não seria observada se tivéssemos implementado nossos algoritmos em uma outra linguagem, como o c++ ou o FORTRAN.

5.4.2 Atualizando a base em todos os subproblemas gerados pelo *Branch-and-Bound*.

Em nosso segundo teste, avaliamos a aplicação contínua da atualização da base. Assim, tomamos dois problemas de programação quadrática inteira mista associados à otimização de carteiras de investimento e aplicamos a atualização sempre que o método de Lemke foi utilizado para resolver um subproblema gerado pelo *Branch-and-Bound*.

Como no teste acima, o primeiro problema que investigamos foi gerado tomando $\lambda = 0{,}9$ para a instância FTSE, enquanto o segundo problema foi obtido tomando $\lambda = 0{,}9$ para a instância S&P. A Tabela 5.12 contém os resultados encontrados.

Tabela 5.12 – Desempenho do *Branch-and-Bound* com e sem o uso da atualização da decomposição LU.

Decomposição LU	Tempo (s)	
	Problema 1	Problema 2
Sem atualização	0,36	4,56
Com atualização	0,51	6,01

A Tabela 5.12 indica que, com relação ao tempo global consumido para resolver o problema de programação quadrática inteira, a vantagem do método sem atualização varia de 24% a 30%. Essa vantagem, naturalmente, é menor que aquela apresentada na Tabela 5.11, uma vez que o número de iterações do método de Lemke em cada nó da árvore não é tão grande, em virtude de aproveitarmos a base ótima do nó pai.

De fato, como o número de fatorações LU feitas em cada nó da árvore costuma ser pequeno, parece-nos que, para o conjunto de testes utilizado neste trabalho, a atualização da base no método de Lemke teria uma pequena influência no desempenho geral do algoritmo, mesmo se fosse implementada de forma eficiente em uma linguagem como o c++. Naturalmente, isso pode não ser verdade para outros problemas de otimização de carteiras de investimento, de modo que esse tópico merece um estudo mais aprofundado.

5.5 COMPARANDO DOIS ALGORITMOS PARA PROGRAMAÇÃO QUADRÁTICA

Uma vez que o subproblema (2.1) nada mais é do que um problema de programação quadrática contínua, vários algoritmos podem ser usados para resolvê-lo. Em especial, optamos pelo método de Lemke pelo fato do mesmo usar a solução (em geral infactível) de um nó pai como ponto de partida para resolver o subproblema de um nó filho correspondente. Uma outra alternativa para aproveitar a solução de um nó pai seria utilizar uma rotina para resolver problemas de programação quadrática que possa começar o processo de otimizaçao a partir de um ponto infactível próximo do ótimo tal como o métode de Lemke.

O programa Matlab possui uma função que resolve problemas de programação quadrática que pode começar o processo de otimização partir de qualquer ponto, seja ele factível ou não. Tal função, denominada *quadprog*, é baseada no método de restrições ativas apresentado por em Gill, P. E *et al.* [9] e pode resolver problemas de tamanho médio (que é o caso dos nossos subproblemas), bem como problemas grandes.

Nesta seção, comparamos o desempenho do nosso algoritmo para programação quadrática inteira mista utilizando tanto o método de Lemke quanto a rotina *quadprog* para resolver os subproblemas associados aos nós da árvore do *Branch-and-Bound*.

Para verificar se o uso da solução do nó pai é capaz de acelerar o processo de otimização da rotina *quadprog*, testamos também o uso dessa função a partir da solução nula $[0, ..., 0]^T$.

Para efeito de notaçao, vamos chamar de *Lemke* a estratégia que consiste em resolver os nós via método de Lemke, *Quadprog 1* a estratégia na qual usamos a função *quadprog*, partindo da solução do nó pai, e de *Quadprog 2* a estratégia na qual *quadprog* usa como ponto inicial a solução nula.

Nos testes dessa seção, utilizamos os mesmos problemas da instância S&P gerados na Seção (5.3). Aproveitando os resultados obtidos na Seção (5.3), ramificamos sempre a variável mais distante de um de seus dois limites, e selecionamos o nó pendente que possui o menor valor de ζ. Como o número de iterações do *Branch-and-Bound* é o mesmo nos 3 casos, comparamos

apenas o tempo consumido pelo programa em cada caso. Os resultados obtidos estão resumidos na Tabela 5.13 e na Figura 5.13.

Tabela 5.13 – Desempenho do algoritmo para a instância S&P, variando a forma de se resolver o Subproblema (2.1)

λ	Tempo(s)		
	Lemke	Quadprog 1	Quadprog 2
0,88	4,12	184,55	140,66
0,89	6,34	264,30	200,08
0,90	7,47	317,10	240,64
0,91	6,06	229,91	188,21
0,92	7,00	208,31	214,38
0,93	11,61	329,34	340,99
0,94	8,77	245,22	255,78
0,95	9,04	238,24	248,76
0,96	7,45	180,17	188,42
0,97	8,42	178,11	186,37
0,98	3,77	72,05	74,90

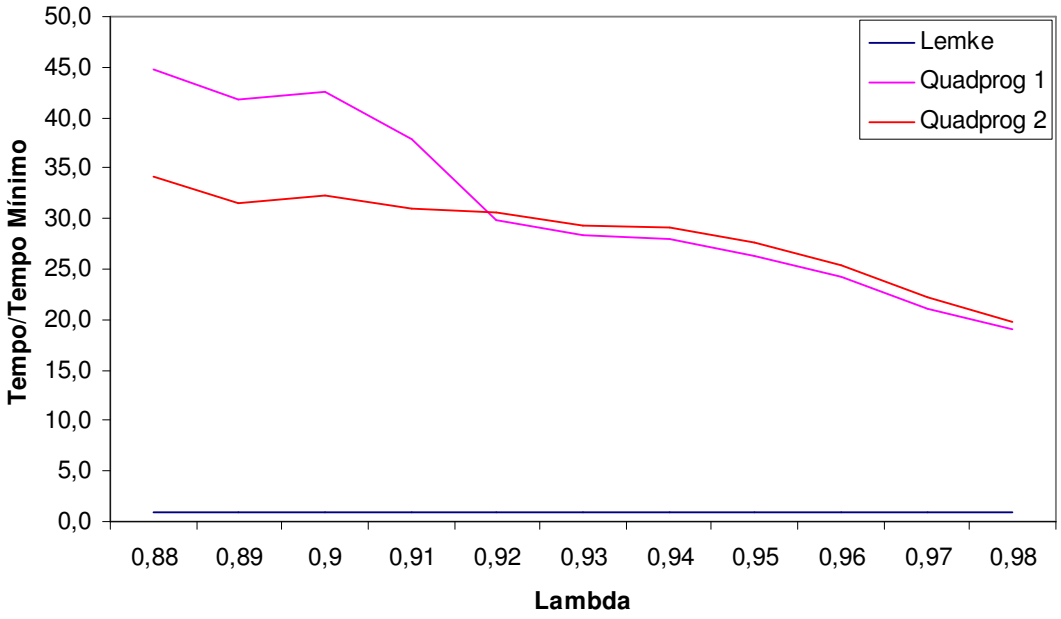


Figura 5.13 – Comparação das estratégias de resolução do subproblema (2.1) para a instância S&P.

Analisando o gráfico e a tabela acima, observamos que o método de Lemke foi de 20 a 45 vezes mais rápido que o *quadprog*. Tais resultados comprovam a efeciência do método de Lemke para resolver problemas quadrátcos partindo de uma "boa solução complementar". Já quanto aos testes distintos feitos sobre a rotina *quadprog*, observamos que, de forma surpreendente, para valores de λ entre 0,88 e 0,91, o processo de otimização foi razoavelmente mais rápido quando partimos da solução nula Já para valores de λ entre 0,92 e 0,98, a escolha do ponto inicial teve pouca influência sobre o desempenho da rotina *quadprog*, havendo ligeira vantagem para a estratégia na qual partimos da solução do nó pai.

5.6 TESTANDO O DESEMPENHO DO PROGRAMA

Nas seções anteriores, determinamos as melhores combinações de estratégias para o *Branch-and-Bound*. Nesta seção, avaliamos o desempenho global de nosso algoritmo aplicando a melhor dessas combinações à solução de um grande número de problemas otimização de carteiras de investimento.

Os problemas gerados têm a forma (1.5) e estão associados às cinco instâncias já mencionadas. Para cada instância, definimos 50 valores igualmente espaçados para λ, de modo que $\lambda \in \left\{0, \frac{1}{49}, \frac{2}{49}, ..., \frac{49}{49}\right\}$. Assim, utilizamos 250 problemas diferentes em nossos testes.

A análise do desempenho de nosso método, neste caso, é pouco ortodoxa, pois comparamos os resultados ora obtidos com aqueles apresentados por Dias [7], que emprega um algoritmo genético para resolver o mesmo problema de otimização de *portfolios*. Nosso objetivo é verificar se é possível encontrar um resultado melhor que o fornecido por uma metaheurística em um tempo computacional pequeno.

Os resultados que obtivemos são apresentados nas Tabelas 5.14 a 5.23. Nessas tabelas, apresentamos o valor escolhido para λ, o número de iterações do algoritmo *Branch-and-*

Bound (Iter), o percentual investido do capital disponível (Gasto), o valor final da função objetivo para o nosso algoritmo (ζ_{BB}), o valor final da função objetivo obtido pelo algoritmo de Dias (ζ_{AG}), a diferença entre estes dois valores ($\zeta_{BB} - \zeta_{AG}$) e a melhora percentual relativa obtida por nosso algoritmo (Melhora), ou seja, o valor $100(\zeta_{BB} - \zeta_{AG})/ \zeta_{AG}$.

A análise das tabelas revela que, para todas as instâncias, o número de nós analisados e o tempo gasto são ínfimos para valores de λ menores que 0,8. Em geral, a solução do nó raiz é a solução do problema e o esforço computacional para obtê-la é mínimo, mostrando que o método é eficiente para casos simples.

Como discutido anteriormente, quando λ é maior que 0,8, priorizamos quase que exclusivamente o risco, o que torna o problema mais complexo, pois há a necessidade de diversificar bastante o *portfolio*. Em nossos testes, as soluções para $0,8 \leq \lambda < 1$ continuam sendo facilmente obtidas para a instância Hang Seng. Já a segunda e a terceira instâncias (DAX e FTSE, respectivamente), exigiram um pouco mais do algoritmo, que fez cerca de 20 iterações e chegou a consumir 0,5 segundos em um problema de FTSE. A instância Nikkei (a última apresentada), apesar de conter um número alto de ativos, não chegou a apresentar dificuldades para nosso algoritmo, que não precisou de mais de 10 iterações e 2,25 segundos para chegar ao ótimo para todos os valores de λ. Finalmente, quarta instância, S&P, se mostrou a mais difícil de resolver, pois, para $\lambda > 0,9$, o tempo de execução superou os 6 segundos, o número de iterações ultrapassou 800 e foram executadas mais de 3100 iterações do método de Lemke por problema. Ainda assim, os resultados foram bastante encorajadores.

De uma forma geral, a análise do tempo de execução indica que nosso método é extremamente eficiente quando a aversão ao risco não é muito grande, mantendo-se competitivo mesmo quando λ é alto, ou seja, permitirmos a inclusão de muitos ativos na carteira.

Infelizmente, a comparação direta de nosso algoritmo com o programa elaborado por Dias não é possível para todo o conjunto de problemas, pois permitimos que apenas uma parte do capital disponível seja investida, enquanto o algoritmo genético por ele proposto exige que todo o capital disponível seja aplicado. Em termos práticos, supomos a existência de um ativo livre de risco (caderneta de poupança, letras do tesouro americano, etc) que seja capaz de absorver o montante não investido nos ativos (com risco) que formam nossa carteira. Isso explica, por

exemplo, a solução que obtemos quando $\lambda = 1$. Neste caso, somos tão avessos ao risco que preferimos aplicar todo o nosso dinheiro no ativo livre de risco, de modo que a solução do problema tem $\zeta = 0$.

Como observamos nas tabelas, para as instâncias DAX, FTSE e S&P, não investimos todo o montante disponível quando $\lambda > 0{,}9$. Já para as instâncias Hang Seng e Nikkei, não aplicamos todo o dinheiro para $\lambda > 0{,}75$. Como essa é justamente a faixa na qual o algoritmo genético tem melhor desempenho, a comparação do tempo gasto pelos programas não é possível.

Ainda assim, podemos comparar os valores obtidos para a função objetivo. Neste caso, as tabelas mostram que, quando λ é menor que 0,8, obtemos uma solução melhor que a fornecida pelo algoritmo genético, sem um gasto computacional expressivo. A vantagem percentual de nosso método sobre o algoritmo de Dias varia entre 1,6 e 4,9% para $\lambda = 0$ e diminui à medida que aumentamos λ. Contudo, a Tabela 5.21 mostra que, para a instância S&P, essa diferença percentual volta a aumentar para $\lambda > 0{,}8$, chegando a 26,2% quando $\lambda = 0{,}898$.

Em conclusão, podemos dizer que o método direto é competitivo, obtendo a solução ótima rapidamente quando a aversão ao risco é baixa, e consumindo um tempo bastante aceitável quando é preciso diversificar acentuadamente o portfólio.

Tabela 5.14 – Desempenho do algoritmo para a instância Hang Seng, com $\lambda \in [0; 0,5)$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,0000	0	3	1,0000	0,0104	-108,6500	-103,5858	-5,0642	4,8889
0,0204	0	3	1,0000	0,0058	-105,4581	-100,6226	-4,8355	4,8056
0,0408	0	3	1,0000	0,0058	-102,2661	-97,6595	-4,6066	4,7170
0,0612	0	3	1,0000	0,0059	-99,0742	-94,6963	-4,3779	4,6231
0,0816	0	3	1,0000	0,0058	-95,8822	-91,7331	-4,1491	4,5230
0,1020	0	3	1,0000	0,0143	-92,6903	-88,7699	-3,9204	4,4163
0,1224	0	3	1,0000	0,0058	-89,4984	-85,8068	-3,6916	4,3023
0,1429	0	3	1,0000	0,0059	-86,3064	-82,8436	-3,4628	4,1799
0,1633	0	3	1,0000	0,0058	-83,1145	-79,8804	-3,2341	4,0486
0,1837	0	3	1,0000	0,0058	-79,9225	-76,9173	-3,0052	3,9070
0,2041	0	3	1,0000	0,0058	-76,7306	-73,9545	-2,7761	3,7539
0,2245	0	3	1,0000	0,0358	-73,5387	-70,9916	-2,5471	3,5879
0,2449	0	3	1,0000	0,0062	-70,3467	-68,0287	-2,3180	3,4074
0,2653	0	3	1,0000	0,0067	-67,1548	-65,0658	-2,0890	3,2105
0,2857	0	3	1,0000	0,0058	-63,9629	-62,1034	-1,8595	2,9942
0,3061	0	3	1,0000	0,0058	-60,7709	-59,1461	-1,6248	2,7471
0,3265	0	3	1,0000	0,0059	-57,5790	-56,1888	-1,3902	2,4741
0,3469	0	4	1,0000	0,0061	-54,3905	-53,2315	-1,1590	2,1772
0,3673	0	4	1,0000	0,0065	-51,2917	-50,2751	-1,0166	2,0220
0,3878	0	4	1,0000	0,0060	-48,3045	-47,3845	-0,9200	1,9416
0,4082	0	4	1,0000	0,0061	-45,4122	-44,5889	-0,8233	1,8465
0,4286	0	4	1,0000	0,0060	-42,6013	-41,8743	-0,7270	1,7362
0,4490	0	5	1,0000	0,0346	-39,8856	-39,2321	-0,6535	1,6659
0,4694	0	5	1,0000	0,0189	-37,2929	-36,6760	-0,6169	1,6821
0,4898	0	5	1,0000	0,0061	-34,8084	-34,2274	-0,5810	1,6976

Tabela 5.15 – Desempenho do algoritmo para a instância Hang Seng, com $\lambda \in (0{,}5;\ 1]$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,5102	0	5	1,0000	0,0065	-32,4191	-31,8743	-0,5448	1,7093
0,5306	0	5	1,0000	0,0061	-30,1140	-29,6055	-0,5085	1,7174
0,5510	0	5	1,0000	0,0061	-27,8837	-27,4108	-0,4729	1,7252
0,5714	0	5	1,0000	0,0062	-25,7203	-25,2841	-0,4362	1,7251
0,5918	0	6	1,0000	0,0062	-23,6195	-23,2171	-0,4024	1,7332
0,6122	0	6	1,0000	0,0062	-21,5856	-21,2090	-0,3766	1,7756
0,6327	0	6	1,0000	0,0063	-19,6133	-19,2644	-0,3489	1,8112
0,6531	0	6	1,0000	0,0062	-17,6968	-17,3795	-0,3173	1,8255
0,6735	0	6	1,0000	0,0343	-15,8310	-15,5451	-0,2859	1,8390
0,6939	0	6	1,0000	0,0194	-14,0114	-13,7572	-0,2542	1,8476
0,7143	0	6	1,0000	0,0188	-12,2342	-12,0115	-0,2227	1,8540
0,7347	0	6	1,0000	0,0188	-10,4957	-10,3044	-0,1913	1,8562
0,7551	0	6	1,0000	0,0188	-8,7928	-8,6352	-0,1576	1,8253
0,7755	0	5	0,9020	0,0194	-7,1947	-6,9999	-0,1948	
0,7959	0	5	0,7990	0,0194	-5,7935	-5,4121	-0,3814	
0,8163	0	5	0,7011	0,0185	-4,5754	-3,8752	-0,7002	
0,8367	0	5	0,6080	0,0062	-3,5270	-2,4532	-1,0738	
0,8571	0	5	0,5193	0,0065	-2,6361	-1,0793	-1,5568	
0,8776	0	5	0,4348	0,0061	-1,8917	0,2285	-2,1202	
0,8980	0	5	0,3541	0,0061	-1,2838	1,4701	-2,7539	
0,9184	0	5	0,2770	0,0062	-0,8034	2,6536	-3,4570	
0,9388	0	5	0,2032	0,0061	-0,4421	3,7601	-4,2022	
0,9592	0	5	0,1326	0,0062	-0,1923	4,7703	-4,9626	
0,9796	2	7	0,0651	0,0375	-0,0471	5,6887	-5,7358	
1,0000	0	0	0,0000	0,0180	0,0000	6,4226	-6,4226	

Tabela 5.16 – Desempenho do algoritmo para a instância DAX, com $\lambda \in [0; 0,5)$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,0000	0	3	1,0000	0,0499	-97,9400	-93,7626	-4,1774	4,4553
0,0204	0	3	1,0000	0,0490	-95,3626	-91,3606	-4,0020	4,3805
0,0408	0	3	1,0000	0,0490	-92,7852	-88,9585	-3,8267	4,3017
0,0612	0	3	1,0000	0,0487	-90,2078	-86,5565	-3,6513	4,2184
0,0816	0	3	1,0000	0,0489	-87,6304	-84,1544	-3,4760	4,1305
0,1020	0	3	1,0000	0,0488	-85,0530	-81,7524	-3,3006	4,0374
0,1224	0	3	1,0000	0,0489	-82,4756	-79,3503	-3,1253	3,9386
0,1429	0	3	1,0000	0,0489	-79,8982	-76,9483	-2,9499	3,8336
0,1633	0	4	1,0000	0,0374	-77,3535	-74,5467	-2,8068	3,7651
0,1837	0	4	1,0000	0,0513	-74,9125	-72,2034	-2,7091	3,7521
0,2041	0	4	1,0000	0,0505	-72,5486	-69,9370	-2,6116	3,7342
0,2245	0	4	1,0000	0,0500	-70,2408	-67,7270	-2,5138	3,7117
0,2449	0	4	1,0000	0,0499	-67,9750	-65,5588	-2,4162	3,6855
0,2653	0	4	1,0000	0,0503	-65,7416	-63,4231	-2,3185	3,6557
0,2857	0	4	1,0000	0,0499	-63,5336	-61,3128	-2,2208	3,6221
0,3061	0	4	1,0000	0,0499	-61,3460	-59,2228	-2,1232	3,5852
0,3265	0	4	1,0000	0,0502	-59,1748	-57,1493	-2,0255	3,5443
0,3469	0	4	1,0000	0,0499	-57,0173	-55,0890	-1,9283	3,5004
0,3673	0	4	1,0000	0,0501	-54,8711	-53,0392	-1,8319	3,4539
0,3878	0	4	1,0000	0,0504	-52,7345	-51,0017	-1,7328	3,3976
0,4082	0	4	1,0000	0,0499	-50,6060	-48,9710	-1,6350	3,3387
0,4286	0	4	1,0000	0,0504	-48,4845	-46,9471	-1,5374	3,2747
0,4490	0	4	1,0000	0,0509	-46,3689	-44,9291	-1,4398	3,2046
0,4694	0	4	1,0000	0,0506	-44,2586	-42,9164	-1,3422	3,1276
0,4898	0	4	1,0000	0,0378	-42,1528	-40,9084	-1,2444	3,0418

Tabela 5.17 – Desempenho do algoritmo para a instância DAX, com $\lambda \in (0{,}5;\ 1]$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,5102	0	5	1,0000	0,0521	-40,0562	-38,9042	-1,1520	2,9611
0,5306	0	5	1,0000	0,0519	-37,9807	-36,9063	-1,0744	2,9111
0,5510	0	5	1,0000	0,0514	-35,9248	-34,9258	-0,9990	2,8604
0,5714	0	5	1,0000	0,0512	-33,8862	-32,9627	-0,9235	2,8017
0,5918	0	5	1,0000	0,0511	-31,8633	-31,0151	-0,8482	2,7346
0,6122	0	5	1,0000	0,0510	-29,8544	-29,0815	-0,7729	2,6576
0,6327	0	5	1,0000	0,0516	-27,8583	-27,1610	-0,6973	2,5672
0,6531	0	5	1,0000	0,0511	-25,8736	-25,2518	-0,6218	2,4625
0,6735	0	5	1,0000	0,0511	-23,8993	-23,3530	-0,5463	2,3394
0,6939	0	5	1,0000	0,0512	-21,9347	-21,4634	-0,4713	2,1960
0,7143	0	5	1,0000	0,0521	-19,9787	-19,5837	-0,3950	2,0168
0,7347	0	6	1,0000	0,0516	-18,0357	-17,7124	-0,3233	1,8255
0,7551	0	6	1,0000	0,0520	-16,1218	-15,8546	-0,2672	1,6854
0,7755	0	6	1,0000	0,0521	-14,2362	-14,0243	-0,2119	1,5110
0,7959	0	6	1,0000	0,0519	-12,3768	-12,2226	-0,1542	1,2618
0,8163	0	7	1,0000	0,0530	-10,5473	-10,3520	-0,1953	1,8865
0,8367	0	7	1,0000	0,0525	-8,7530	-8,6964	-0,0566	0,6505
0,8571	0	8	1,0000	0,0537	-7,0031	-6,9850	-0,0181	0,2588
0,8776	0	10	1,0000	0,0554	-5,3410	-5,3371	-0,0039	0,0728
0,8980	20	47	1,0000	0,1701	-3,7934	-3,6992	-0,0942	2,5461
0,9184	20	53	0,8991	0,1920	-2,3982	-2,2798	-0,1184	
0,9388	20	53	0,6596	0,1869	-1,3196	-1,0382	-0,2814	
0,9592	20	53	0,4304	0,1873	-0,5740	-0,0140	-0,5600	
0,9796	22	62	0,3142	0,2212	-0,1404	0,9299	-1,0703	
1,0000	0	0	0,0000	0,0353	0,0000	1,5926	-1,5926	

Tabela 5.18 – Desempenho do algoritmo para a instância FTSE, com $\lambda \in [0; 0,5)$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,0000	0	3	1,0000	0,0550	-82,0900	-79,2322	-2,8578	3,6069
0,0204	0	3	1,0000	0,0545	-80,1052	-77,3492	-2,7560	3,5631
0,0408	0	3	1,0000	0,0546	-78,1204	-75,4662	-2,6542	3,5171
0,0612	0	3	1,0000	0,0532	-76,1355	-73,5832	-2,5523	3,4686
0,0816	0	3	1,0000	0,0544	-74,1507	-71,7002	-2,4505	3,4177
0,1020	0	3	1,0000	0,0544	-72,1659	-69,8172	-2,3487	3,3640
0,1224	0	3	1,0000	0,0554	-70,1811	-67,9342	-2,2469	3,3074
0,1429	0	3	1,0000	0,0535	-68,1962	-66,0512	-2,1450	3,2474
0,1633	0	3	1,0000	0,0543	-66,2114	-64,1682	-2,0432	3,1841
0,1837	0	3	1,0000	0,0532	-64,2266	-62,2852	-1,9414	3,1169
0,2041	0	3	1,0000	0,0547	-62,2418	-60,4022	-1,8396	3,0455
0,2245	0	3	1,0000	0,0418	-60,2569	-58,5192	-1,7377	2,9694
0,2449	0	3	1,0000	0,0565	-58,2721	-56,6362	-1,6359	2,8884
0,2653	0	3	1,0000	0,0531	-56,2873	-54,7532	-1,5341	2,8018
0,2857	0	3	1,0000	0,3701	-54,3025	-52,8702	-1,4323	2,7090
0,3061	0	3	1,0000	0,0535	-52,3176	-50,9872	-1,3304	2,6092
0,3265	0	3	1,0000	0,0543	-50,3328	-49,1042	-1,2286	2,5019
0,3469	0	3	1,0000	0,0543	-48,3480	-47,2212	-1,1268	2,3861
0,3673	0	3	1,0000	0,0428	-46,3632	-45,3383	-1,0249	2,2607
0,3878	0	3	1,0000	0,0657	-44,3784	-43,4553	-0,9231	2,1244
0,4082	0	3	1,0000	0,0535	-42,3935	-41,5723	-0,8212	1,9755
0,4286	2	6	1,0000	0,0685	-40,4087	-39,6893	-0,7194	1,8127
0,4490	2	7	1,0000	0,0693	-38,4365	-37,8063	-0,6302	1,6670
0,4694	0	5	1,0000	0,0561	-36,4911	-35,9246	-0,5665	1,5769
0,4898	0	5	1,0000	0,0564	-34,5749	-34,0623	-0,5126	1,5048

Tabela 5.19 – Desempenho do algoritmo para a instância FTSE, com $\lambda \in (0{,}5;\ 1]$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,5102	0	5	1,0000	0,0556	-32,6846	-32,2260	-0,4586	1,4230
0,5306	0	5	1,0000	0,0551	-30,8171	-30,4146	-0,4025	1,3234
0,5510	0	5	1,0000	0,0553	-28,9699	-28,6234	-0,3465	1,2105
0,5714	2	8	1,0000	0,0694	-27,1409	-26,8504	-0,2905	1,0819
0,5918	0	7	1,0000	0,0570	-25,3396	-25,0936	-0,2460	0,9804
0,6122	0	7	1,0000	0,0568	-23,5793	-23,3591	-0,2202	0,9425
0,6327	0	7	1,0000	0,0576	-21,8564	-21,6613	-0,1951	0,9005
0,6531	0	7	1,0000	0,0564	-20,1675	-19,9975	-0,1700	0,8503
0,6735	0	7	1,0000	0,0568	-18,5095	-18,3654	-0,1441	0,7847
0,6939	2	10	1,0000	0,0702	-16,8796	-16,7567	-0,1229	0,7336
0,7143	0	8	1,0000	0,0582	-15,2770	-15,1859	-0,0911	0,5996
0,7347	0	8	1,0000	0,0577	-13,6999	-13,6343	-0,0656	0,4813
0,7551	0	8	1,0000	0,0577	-12,1463	-12,1056	-0,0407	0,3362
0,7755	0	9	1,0000	0,0585	-10,6195	-10,6081	-0,0114	0,1073
0,7959	2	12	1,0000	0,0718	-9,1297	-9,1217	-0,0080	0,0877
0,8163	22	55	1,0000	0,1899	-7,6860	-7,6849	-0,0011	0,0149
0,8367	20	40	1,0000	0,1746	-6,2983	-6,2979	-0,0004	0,0064
0,8571	20	47	1,0000	0,1861	-4,9709	-4,9229	-0,0480	0,9750
0,8776	20	44	1,0000	0,1849	-3,7009	-3,6048	-0,0961	2,6668
0,8980	22	51	0,8789	0,2048	-2,5225	-2,4342	-0,0883	
0,9184	22	51	0,6875	0,2023	-1,5786	-1,2563	-0,3223	
0,9388	22	51	0,5044	0,2040	-0,8686	-0,2279	-0,6407	
0,9592	22	51	0,3291	0,2032	-0,3778	0,7917	-1,1695	
0,9796	62	127	0,1649	0,5002	-0,0921	1,6221	-1,7142	
1,0000	0	0	0,0000	0,0382	0,0000	2,3609	-2,3609	

Tabela 5.20 – Desempenho do algoritmo para a instância S&P, com $\lambda \in [0; 0,5)$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,0000	0	3	1,0000	0,0651	-91,9500	-89,5664	-2,3836	2,6613
0,0204	0	3	1,0000	0,0623	-89,4737	-87,2158	-2,2579	2,5889
0,0408	0	3	1,0000	0,0636	-86,9975	-84,8652	-2,1323	2,5126
0,0612	0	3	1,0000	0,0626	-84,5212	-82,5146	-2,0066	2,4318
0,0816	0	3	1,0000	0,0621	-82,0449	-80,1640	-1,8809	2,3464
0,1020	0	3	1,0000	0,0620	-79,5686	-77,8134	-1,7552	2,2557
0,1224	0	4	1,0000	0,0643	-77,1130	-75,4650	-1,6480	2,1838
0,1429	0	4	1,0000	0,0644	-74,7137	-73,1556	-1,5581	2,1298
0,1633	0	4	1,0000	0,0639	-72,3516	-70,8833	-1,4683	2,0714
0,1837	0	5	1,0000	0,0656	-70,0151	-68,6359	-1,3792	2,0095
0,2041	0	5	1,0000	0,0672	-67,7140	-66,4083	-1,3057	1,9661
0,2245	0	5	1,0000	0,0667	-65,4452	-64,2098	-1,2354	1,9240
0,2449	0	5	1,0000	0,0655	-63,2007	-62,0355	-1,1652	1,8783
0,2653	0	5	1,0000	0,0658	-60,9749	-59,8798	-1,0951	1,8288
0,2857	0	5	1,0000	0,0646	-58,7636	-57,7387	-1,0249	1,7750
0,3061	0	5	1,0000	0,0663	-56,5641	-55,6094	-0,9547	1,7168
0,3265	0	5	1,0000	0,0650	-54,3742	-53,4896	-0,8846	1,6538
0,3469	0	5	1,0000	0,0656	-52,1920	-51,3785	-0,8135	1,5834
0,3673	0	5	1,0000	0,0652	-50,0164	-49,2735	-0,7429	1,5077
0,3878	0	5	1,0000	0,0672	-47,8464	-47,1736	-0,6728	1,4262
0,4082	0	6	1,0000	0,0659	-45,6846	-45,0809	-0,6037	1,3392
0,4286	0	6	1,0000	0,0662	-43,5574	-42,9977	-0,5597	1,3017
0,4490	0	6	1,0000	0,0666	-41,4655	-40,9465	-0,5190	1,2676
0,4694	0	6	1,0000	0,0663	-39,4043	-38,9259	-0,4784	1,2289
0,4898	0	6	1,0000	0,0544	-37,3700	-36,9321	-0,4379	1,1856

Tabela 5.21 – Desempenho do algoritmo para a instância S&P, com $\lambda \in (0{,}5;\ 1]$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,5102	0	6	1,0000	0,0682	-35,3592	-34,9630	-0,3962	1,1331
0,5306	0	6	1,0000	0,0667	-33,3694	-33,0170	-0,3524	1,0672
0,5510	0	6	1,0000	0,0669	-31,3982	-31,0911	-0,3071	0,9877
0,5714	0	6	1,0000	0,0657	-29,4435	-29,1815	-0,2620	0,8979
0,5918	0	6	1,0000	0,0661	-27,5038	-27,2869	-0,2169	0,7950
0,6122	0	6	1,0000	0,0665	-25,5774	-25,4059	-0,1715	0,6748
0,6327	0	6	1,0000	0,0665	-23,6631	-23,5378	-0,1253	0,5321
0,6531	0	7	1,0000	0,0674	-21,7674	-21,6836	-0,0838	0,3863
0,6735	4	13	1,0000	0,0989	-19,8996	-19,8500	-0,0496	0,2499
0,6939	0	10	1,0000	0,0705	-18,0735	-18,0430	-0,0305	0,1688
0,7143	0	10	1,0000	0,0705	-16,3050	-16,2821	-0,0229	0,1409
0,7347	0	10	1,0000	0,0705	-14,5905	-14,5752	-0,0153	0,1052
0,7551	2	13	1,0000	0,0879	-12,9262	-12,9179	-0,0083	0,0642
0,7755	0	11	1,0000	0,0589	-11,3161	-11,3110	-0,0051	0,0448
0,7959	0	11	1,0000	0,0741	-9,7588	-9,7571	-0,0017	0,0172
0,8163	0	14	1,0000	0,0750	-8,2528	-8,0098	-0,2430	3,0335
0,8367	26	85	1,0000	0,2548	-6,7970	-5,5060	-1,2910	23,4463
0,8571	32	109	1,0000	0,3067	-5,4069	-5,1612	-0,2457	4,7614
0,8776	96	267	1,0000	0,6938	-4,0954	-3,6679	-0,4275	11,6553
0,8980	578	2274	1,0000	3,9129	-2,8571	-2,2624	-0,5947	26,2852
0,9184	826	3126	0,8039	6,1316	-1,7890	-1,4043	-0,3847	
0,9388	826	3126	0,5898	6,1043	-0,9844	-0,6276	-0,3568	
0,9592	826	3126	0,3848	6,2236	-0,4282	0,2225	-0,6507	
0,9796	804	3191	0,1884	6,5174	-0,1048	0,9968	-1,1016	
1,0000	0	0	0	0,0597	0,0000	1,5094	-1,5094	

Tabela 5.22 – Desempenho do algoritmo para a instância Nikkei, com $\lambda \in [0; 0,5)$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,0000	0	3	1,0000	0,5192	-39,7100	-39,0365	-0,6735	1,7253
0,0204	0	3	1,0000	0,5227	-38,5632	-37,9349	-0,6283	1,6562
0,0408	0	3	1,0000	0,5271	-37,4163	-36,8334	-0,5829	1,5827
0,0612	0	3	1,0000	0,5213	-36,2695	-35,7318	-0,5377	1,5049
0,0816	0	3	1,0000	0,5177	-35,1226	-34,6302	-0,4924	1,4219
0,1020	0	3	1,0000	0,5146	-33,9758	-33,5286	-0,4472	1,3337
0,1224	0	4	1,0000	0,5365	-32,8326	-32,4271	-0,4055	1,2506
0,1429	0	4	1,0000	0,5404	-31,7246	-31,3434	-0,3812	1,2162
0,1633	0	4	1,0000	0,5394	-30,6465	-30,2894	-0,3571	1,1789
0,1837	0	4	1,0000	0,5335	-29,5883	-29,2555	-0,3328	1,1377
0,2041	0	4	1,0000	0,5401	-28,5440	-28,2353	-0,3087	1,0933
0,2245	0	4	1,0000	0,5344	-27,5098	-27,2253	-0,2845	1,0450
0,2449	0	4	1,0000	0,5219	-26,4832	-26,2231	-0,2601	0,9917
0,2653	0	4	1,0000	0,5502	-25,4624	-25,2266	-0,2358	0,9348
0,2857	0	4	1,0000	0,5313	-24,4463	-24,2347	-0,2116	0,8730
0,3061	0	5	1,0000	0,5462	-23,4344	-23,2465	-0,1879	0,8084
0,3265	0	6	1,0000	0,5565	-22,4335	-22,2614	-0,1721	0,7732
0,3469	0	6	1,0000	0,5498	-21,4525	-21,2893	-0,1632	0,7666
0,3673	0	6	1,0000	0,5454	-20,4885	-20,3343	-0,1542	0,7585
0,3878	0	6	1,0000	0,5551	-19,5389	-19,3936	-0,1453	0,7494
0,4082	2	9	1,0000	0,8937	-18,6014	-18,4651	-0,1363	0,7383
0,4286	0	7	1,0000	0,5506	-17,6758	-17,5470	-0,1288	0,7340
0,4490	0	7	1,0000	0,5647	-16,7611	-16,6387	-0,1224	0,7357
0,4694	0	7	1,0000	0,5568	-15,8559	-15,7398	-0,1161	0,7375
0,4898	0	7	1,0000	0,5551	-14,9589	-14,8492	-0,1097	0,7386

Tabela 5.23 – Desempenho do algoritmo para a instância Nikkei, com $\lambda \in (0{,}5;\ 1]$.

λ	Iter	ItLemke	Gasto	Tempo (s)	ζ_{BB} ($x10^{-4}$)	ζ_{AG} ($x10^{-4}$)	$\zeta_{BB} - \zeta_{AG}$ ($x10^{-4}$)	Melhora (%)
0,5102	0	7	1,0000	0,5649	-14,0692	-13,9659	-0,1033	0,7395
0,5306	0	7	1,0000	0,5647	-13,1860	-13,0891	-0,0969	0,7406
0,5510	0	7	1,0000	0,5579	-12,3085	-12,2179	-0,0906	0,7412
0,5714	0	7	1,0000	0,5667	-11,4361	-11,3519	-0,0842	0,7414
0,5918	0	7	1,0000	0,5593	-10,5683	-10,4905	-0,0778	0,7415
0,6122	0	8	1,0000	0,5499	-9,7059	-9,6339	-0,0720	0,7477
0,6327	0	8	1,0000	0,5704	-8,8497	-8,7827	-0,0670	0,7624
0,6531	0	8	1,0000	0,5647	-7,9992	-7,9373	-0,0619	0,7802
0,6735	0	8	1,0000	0,5667	-7,1538	-7,0969	-0,0569	0,8014
0,6939	2	11	1,0000	0,9372	-6,3132	-6,2637	-0,0495	0,7907
0,7143	0	9	1,0000	0,5786	-5,4788	-5,4361	-0,0427	0,7864
0,7347	0	9	1,0000	0,5751	-4,6508	-4,6144	-0,0364	0,7891
0,7551	0	8	0,9184	0,5703	-3,8575	-3,7988	-0,0587	
0,7755	0	8	0,8197	0,5842	-3,1561	-2,9911	-0,1650	
0,7959	2	10	0,7261	0,9065	-2,5415	-2,1883	-0,3532	
0,8163	2	10	0,6371	0,9063	-2,0071	-1,3904	-0,6167	
0,8367	2	10	0,5525	0,9052	-1,5471	-0,6103	-0,9368	
0,8571	2	10	0,4719	0,9111	-1,1563	0,1336	-1,2899	
0,8776	2	10	0,3950	0,9089	-0,8296	0,9597	-1,7893	
0,8980	2	10	0,3219	0,8961	-0,5630	1,5516	-2,1146	
0,9184	2	10	0,2518	0,9098	-0,3523	2,3262	-2,6785	
0,9388	4	12	0,1852	1,2253	-0,1938	2,6550	-2,8488	
0,9592	6	14	0,1195	1,5317	-0,0842	2,9427	-3,0269	
0,9796	10	22	0,0594	2,2467	-0,0204	3,1447	-3,1651	
1,0000	0	0	0,0000	0,4963	0,0000	3,4144	-3,4144	

5.7 RESOLVENDO PROBLEMAS COM MAIS RESTRIÇÕES

Nos testes anteriores, trabalhamos com um problema quadrático que continha apenas a restrição de limite do capital gasto, uma vez que a restrição de cardinalidade e as restrições que relacionam as variáveis inteiras e reais foram tratadas de forma implícita[7]. Nesta seção, aplicamos nosso algoritmo à resolução do problema original na forma (1.3), ou seja, incluindo uma restrição explícita associada ao retorno do *portfolio*. Entretanto, como só trabalhamos com restrições de desigualdade, essa nova restrição foi escrita na forma $\mu^T x \geq \rho$, onde ρ é o retorno mínimo desejado. Temos, assim, um problema com duas restrições, mas sem o parâmetro de penalização λ. Deve-se observar que, com essa alteração, a função objetivo perde o termo linear. Além disso, como descrito na Seção 1.6, não consideramos as restrições de limite superior.

Para facilitar a geração dos problemas, definimos o retorno mínimo, ρ, como uma porcentagem do retorno máximo, ρ_{max}, que se pode obter com a carteira, ou seja, ignorando-se completamente o risco e investindo-se todo o capital disponível no ativo de maior retorno. As porcentagens de retorno adotadas foram iguais a 40% e 50% de ρ_{max}. Os resultados obtidos.são mostrados na Tabela 5.24.

A análise da tabela indica que o método apresentou bom desempenho quando aplicado a esse problema. Para a instância Hang Seng, o tempo gasto foi muito baixo, comportamento similar àquele observado na Seção 5.6. Para as instâncias DAX, FTSE e S&P, o número de iterações oscilou entre 30 e 70 e o número de iterações do método de Lemke ficou entre 100 e 225, mas o tempo consumido permaneceu inferior a um segundo, sugerindo que tanto o método de Lemke como o *Branch-and-Bound* não foram substancialmente afetados pela inclusão de uma restrição. Finalmente, a instância Nikkei exigiu um tempo de execução superior a um segundo, mas com um número muito baixo de iterações, como nos testes feitos anteriormente.

[7] Cabe lembrar que, apesar do problema quadrático (2.1) possuir apenas uma restrição, o método de Lemke trabalha com a formulação (2.2), que envolve $N + 1$ restrições, excluindo as de complementaridade.

Tabela 5.24 – Desempenho do algoritmo quando aplicado ao problema (1.3).

Instância	ρ	Gasto	Iter	ItLemke	Tempo
Hang Seng	0,4 ρ_{max}	0,6116	0	6	0,0088
	0,5 ρ_{max}	0,7645	0	6	0,0336
DAX	0,4 ρ_{max}	0,5407	49	124	0,4219
	0,5 ρ_{max}	0,6759	49	128	0,4149
FTSE	0,4 ρ_{max}	0,4000	71	134	0,5094
	0,5 ρ_{max}	0,6232	50	100	0,4931
S&P	0,4 ρ_{max}	1,0000	64	225	0,9648
	0,5 ρ_{max}	1,0000	31	134	0,6299
Nikkei	0,4 ρ_{max}	0,4713	3	11	1,1090
	0,5 ρ_{max}	0,5891	3	11	1,0724

5.8 USANDO RESTRIÇÕES DE IGUALDADE

Como já foi dito, em todos os testes que efetuamos, trabalhamos apenas com restrições de desigualdade. Naturalmente, seria possível converter uma restrição de igualdade em duas restrições de desigualdade, de modo a permitir que o método de Lemke resolvesse qualquer problema de programação quadrática. Apesar de essa conversão acarretar um gasto adicional provocado pelo aumento do problema, esse gasto seria pequeno se comparado ao trabalho computacional já investido na resolução do problema.

Entretanto, essa alteração traz duas dificuldades para o nosso algoritmo. A primeira é que ela pode aumentar o número de ativos na carteira, o que implicaria em um aumento do número de nós na árvore gerada pelo *Branch-and-Bound*. Assim, para valores altos de λ, nosso método poderia consumir um tempo superior ao de um método heurístico eficiente. Podemos dizer, portanto, que nosso método é mais eficiente quando existe um ativo livre de risco, como visto na Seção 5.6, pois os métodos heurísticos não são capazes de se beneficiar dessa característica do modelo.

Um segundo problema observado na conversão de uma restrição de igualdade em duas desigualdades diz respeito à ocorrência de ciclagem no método de Lemke, em decorrência da existência de duas colunas linearmente dependentes na matriz do problema. Neste caso, é preciso adotar alguma estratégia para evitar que uma mesma base venha a ser visitada mais de uma vez durante a execução do método, algo que pretendemos investigar em um trabalho futuro.

CONCLUSÕES

Analisando os resultados obtidos como um todo, é possível observar que a combinação do método de Lemke com o *Branch-and-Bound*, quando bem implementada, apresenta um bom desempenho para resolver problemas quadráticos baseados no modelo de Markowitz para o problema de carteiras de ações. Mais que isso, nos casos em que desejamos montar uma carteira com um número relativamente pequeno de ativos, o método exato consegue fornecer a solução ótima em um tempo comparável ao gasto pelas metaheurísticas mais eficientes.

Outro resultado importante de nosso trabalho diz respeito ao método de Lemke. Apesar de pouco utilizado nos dias de hoje, este método mostrou-se extremamente útil quando é preciso resolver um PCL partindo de uma base inicial boa e infactível. Sendo assim, ele pode ser considerado como uma alternativa eficiente para a resolução de outros problemas de programação quadrática inteira mista, particularmente quando se aplica um método exato.

Apesar dos bons resultados obtidos, muitas melhorias podem ser incorporadas ao nosso trabalho. Dentre elas, as que pretendemos investigar em um futuro próximo são

- A determinação de novos vetores de cobertura, h, capazes de acelerar ainda mais a convergência do método de Lemke.
- A implementação do programa em uma linguagem de alto nível, tal como o c ou o c++. Isso não só tornaria mais rápido o algoritmo, como nos permitiria aproveitar eficientemente a estratégia de atualização da fatoração LU.

- A correção do problema de ciclagem apresentado pelo método de Lemke, de modo a aumentar a robustez do algoritmo quando aplicado a problemas com restrições de igualdade.
- O uso da busca em largura aliada à computação paralela, o que nos permitiria acelerar a convergência ao longo do processo do *Branch-and-Bound*.
- A aplicação de nosso método à solução de problemas reais de programação quadrática inteiro-mista que não possuam relação com o mercado financeiro.

REFERÊNCIAS

[1] BARTELS, R. H. A stabilization of the simplex method. *Numerische Mathematik*, 16 (5), p. 414-434, 1971.

[2] BAZARAA, M. S; SHERALI, H. D; SHETTY, C. M. *Nonlinear programming: theory and algorithms*. New York: John Wiley & Sons Inc, 1979.

[3] BERTSIMAS, D; SHIODA, R. An Algorithm for cardinality constrained quadratic optimization. Por aparecer em *Computational Optimization and Applications*.

[4] BIENSTOCK, D. Computacional study of a family of mixed-integer quadratic programming problems. *Mathematical Programming,* 74, p. 121-140, 1996

[5] CHANG, T. J.; MEADE, N.; BEASLEY, J. E.; SHARAIHA, Y. M. Heuristics for cardinality constrained portfolio optimization. *Computer and Operations Research,* 27, p. 1271–1302, 2000.

[6] COTTLE, R. W; PANG, J. S; STONE, R. E. *The Linear Complementarity Problem*. San Diego: Academic Press Inc, 1992.

[7] DIAS, C. H. *Um novo algoritmo genético para a otimização de carteiras de investimento com restrições de cardinalidade*. Dissertação de Mestrado em Matemática Aplicada. Departamento de Matemática Aplicada, Universidade de Campinas, Campinas, Brasil, 2008.

[8] FORREST, J. J. H; TOMLIN, J. A. Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming*, 2, p 263–278, 1972.

[9] GILL, P. E; MURRAY, W; WRIGHT, M. H. *Practical Optimization*. London: Academic Press, 1981.

[10] KOBERSTEIN, A. *The Dual Simplex Method, Technics for a Fast and Stable Implementation.* Tese de Doutorado em Sistemas Dinâmicos de Inteligência.

Departamento de Ciência da Computação, Universidade de Wirschaftswissenschaften, Paderborn, Alemanha, 2005.

[11] LEMKE, C.E. J. J. H; TOMLIN, J. Bimatrix Equilibrium Points and Mathematical Programming. *Management Science*, 11, p 681-689, 1965.

[12] LUENBERGER, D. G. *Investment Science*. New York: Oxford University press, 1998.

[13] MARKOWITZ, H. Portfolio Selection. *Journal of Finance*, 7, p. 77-91, 1952.

[14] MITRA, G.; KYRIAKIS, T.; LUCAS, C. A review of portfolio planning: Models and systems. Em: *Advances in Portfolio Construction and Implementation.* S.E. Satchell and A.E. Scowcroft (Eds.), Oxford: Butterworth & Heinmann, 2003, p. 1-39.

[15] MORGAN, S. *A Comparison of Simplex Methods Algorithms.* Dissertação de Mestrado em Ciências. Department of Computer & Information Science & Engeneering, University of Florida, Florida, EUA, 1997.

[16] WOLSEY, L. *Integer Programming.* New York: Wiley, 1998.